

Computer Science and Software Engineering
College of Engineering, University of Canterbury

Master's Thesis

**Evaluation of and Mitigation against
Malicious Traffic in SIP-based VoIP
Applications in a Broadband
Internet Environment**

Tobias Wulff

A thesis submitted in partial fulfilment of the requirements
for the Degree of Master of Science in Computer Science at
the University of Canterbury, New Zealand

Supervisors:

Ray Hunt

Malcolm Shore

Voice Over IP (VoIP) telephony is becoming widespread, and is often integrated into computer networks. Because of this, it is likely that malicious software will threaten VoIP systems the same way traditional computer systems have been attacked by viruses, worms, and other automated agents. While most users have become familiar with email spam and viruses in email attachments, spam and malicious traffic over telephony currently is a relatively unknown threat. VoIP networks are a challenge to secure against such malware as much of the network intelligence is focused on the edge devices and access environment.

A novel security architecture is being developed which improves the security of a large VoIP network with many inexperienced users, such as non-IT office workers or telecommunication service customers. The new architecture establishes interaction between the VoIP backend and the end users, thus providing information about ongoing and unknown attacks to all users. An evaluation of the effectiveness and performance of different implementations of this architecture is done using virtual machines and network simulation software to emulate vulnerable clients and servers through providing apparent attack vectors.

Contents

List of Figures	IX
------------------------	-----------

List of Tables	XI
-----------------------	-----------

1. Introduction	1
1.1. Motivation	1
1.1.1. Security Architectures	3
1.1.2. VoIP	4
1.2. Network and Information Security	5
1.3. Scope	6
1.4. Outline	6
2. Voice over IP (VoIP)	9
2.1. History	11
2.2. Components	12
2.2.1. User Agents	12
2.2.2. Servers	14
2.2.3. Session Border Controller (SBC)	15
2.3. Protocols	16
2.3.1. Session Initiation Protocol (SIP)	16
2.3.2. Session Description Protocol (SDP)	18
2.3.3. Real-time Transport Protocol (RTP)	19
2.3.4. Session Traversal Utilities for NAT (STUN)	20
2.4. Security	21
2.4.1. VoIP Applications	21
2.4.2. VoIP Networks	22
3. Attacks and Countermeasures	25
3.1. Introduction	25
3.2. Social Threats	25
3.2.1. VoIP Phishing (Vishing)	26
3.2.2. Rerouting and Man-in-the-Middle Attacks	27
3.3. Technical Threats	27
3.3.1. Denial of Service	27
3.3.2. Circumventing Password Authentication	28
3.3.3. Rogue SIP devices	28

3.3.4.	Kernel-mode and Firmware Rootkits	29
3.3.5.	Fuzzing	30
3.3.6.	Malicious Traffic in the SIP Message Body	30
4.	Related Work	33
4.1.	VoIP Attack Surface	33
4.2.	Protocol Improvements	34
4.3.	Attack Countermeasures	34
4.3.1.	Denial of Service and Flooding Attacks	34
4.3.2.	Eavesdropping	35
4.3.3.	End-user Protection	35
4.4.	Event Correlation and Anomaly Detection	36
4.5.	VoIP Network Simulation	37
5.	Tools	39
5.1.	Testbed Setup	39
5.1.1.	VMware Workstation	40
5.1.2.	Virtual Machines	40
5.2.	Simulation	40
5.2.1.	OMNeT++	41
5.2.2.	INET	42
5.3.	Traffic Generator	42
5.4.	Scanning and Enumeration	43
5.5.	VoIP Infrastructure	44
5.5.1.	Asterisk	44
5.5.2.	SIP Phones	45
5.6.	Honeypots and Honeynets	45
5.6.1.	Dionaea	46
5.6.2.	VoIP Module	47
5.7.	Traffic Analysis	48
5.8.	Event Correlation	49
5.9.	Software Versions	50
5.10.	Unsuitable Tools	51
5.10.1.	Security Information and Event Management (SIEM)	51
5.10.2.	Hardware Phones	52
5.10.3.	Commercial Security Software and Appliances	52
6.	A Novel VoIP Security Architecture	53
6.1.	Motivation	53
6.2.	Concept	54
6.2.1.	End-Users	56
6.2.2.	NAT	57
6.2.3.	Detecting VoIP Phone Crashes	57
6.2.4.	User Notification	60

6.3.	Privacy Concerns	62
6.4.	Scenarios	62
6.4.1.	Scenario 1: SIP Scans	62
6.4.2.	Scenario 2: Phone Crash	63
6.4.3.	Scenario 3: User Reports	65
7.	Implementation	67
7.1.	Proof-of-Concept Testbed	67
7.1.1.	Testbed Security	67
7.1.2.	Collect IDS Data at Residential Gateway	68
7.1.3.	Securely Send Collected Data to Service Provider	69
7.1.4.	User Notification on Different Phones	69
7.1.5.	Event Correlation Rules	70
7.1.6.	Phone Crashes and Phones Unregistering	71
7.2.	Network Simulation	74
7.2.1.	Limitations and Assumptions	75
7.2.2.	Implementation Steps in OMNeT++	76
8.	Evaluation	83
8.1.	Evaluation Process and Input Data	83
8.2.	Honeypot Results	84
8.2.1.	Similarities	86
8.2.2.	SIP Calls	86
8.2.3.	Conclusion	87
8.3.	Security Improvements of the Security Architecture	87
8.3.1.	Detection and Mitigation of SIP Scanning Attacks	88
8.3.2.	Detection and Mitigation of Exploit Attacks	91
8.3.3.	Discussion of RGW and SEC Parameters	93
8.4.	Performance	95
8.5.	Known Problems	96
8.6.	SIP Request and Response Example	98
9.	Conclusion	99
9.1.	Summary	99
9.2.	Future Work	100
9.2.1.	Performance	100
9.2.2.	Realistic VoIP Data	101
9.2.3.	SIP and RTP Attacks Beyond Primitive Scans	101
9.2.4.	Extensions and Generalisation	101
9.2.5.	Incorporating Asterisk	102
	Bibliography	103
	A. SEC Rules	112

A.1. Scan Attack Correlation	112
A.2. Crash Correlation	113
B. Proof-of-concept SIP Notification Scapy Script (Python)	114
C. Simulation Code (C++)	115
C.1. User Agent Module	115
C.2. Residential Gateway Module	117
C.3. Event Correlation Module	119
D. Heartbeat Monitor	122
E. Honeypot	125
E.1. Configuration	125
E.2. Relevant VoIP Module Code (Python)	125

List of Figures

1.1.	Concept of VoIP	2
1.2.	VoIP scans	3
2.1.	VoIP network protocol stack	10
2.2.	Role of SIP and RTP protocols	11
2.3.	Hardware phones	13
2.4.	Softphones	13
2.5.	VoIP conversation using SIP	17
2.6.	VoIP attack vectors	22
3.1.	Popular SSH passwords	29
5.1.	OMNeT++ network simulation graphical user interface	41
5.2.	Scapy sending an incomplete SIP packet	43
5.3.	UML class diagram of SIP honeypot module	48
6.1.	Security architecture inputs	54
6.2.	Heartbeat monitor	58
6.3.	User notification in KPhone	61
6.4.	Scenario 1: Reaction to a SIP OPTIONS scan	63
6.5.	Scenario 2: Reaction to an exploit attack that crashes the phone	64
6.6.	Scenario 3: Reaction to a SPIT call	65
7.1.	Two event correlation examples	70
7.2.	Event correlation in SEC: crash detection	72
7.3.	Event correlation in SEC: scan detection	73
7.4.	Small simulation	74
7.5.	UML class diagram of simulation modules	80
7.6.	Sequence of screenshots of a simulated OPTIONS scan attack	81
8.1.	Number of hosts connecting to VoIP honeypot over time	84
8.2.	Frequency of connections per day to VoIP honeypot	85
8.3.	Section of simulation event log showing one OPTIONS scan	89
8.4.	Section of simulation event log showing blacklist update	90
8.5.	Scan attacks over time, one attacker	91
8.6.	Scan attacks over time, multiple attackers	92
8.7.	Scan attacks over time, multiple attackers, large network	93

8.8. Exploit attacks and signature hits over time, multiple attackers	94
8.9. Comparison of simulation runs with different parameter settings	95

List of Tables

2.1. Important SIP headers	17
2.2. Important SDP headers	19
3.1. VoIP attack methods	26
5.1. Versions of software used for this thesis	51
6.1. Results of OPTIONS request against different phones	60
7.1. Support for SIP MESSAGE request	70
7.2. Round-trip times for different servers from private ADSL in New Zealand (10 pings with a packet size of 64 bytes per target)	75
7.3. SIP message definition for the network simulation	78
7.4. Attack message definition for the network simulation	78
7.5. Report message definition for the network simulation	79
7.6. Update message definition for the network simulation	79
8.1. Honeypot connections by country (percentages rounded)	85
8.2. Notable characteristics of connecting SIP clients sending OPTIONS re- quests	86
8.3. Parameters used for heartbeat interval and phone crash reports threshold and time of first successfully matched exploit signature	94

1. Introduction

1.1. Motivation

Most households in industrialised countries started to use more advanced technology than the old analogue telephones for communication many years ago. Even though cell phone reception tends to be very good in most metropolitan areas, most people prefer a land line phone or equivalent technology because of the higher costs of cell phone calls. With the increasing availability of affordable broadband Internet and flatrate plans it is possible to use more applications over the Internet's IP packet-based communication. This way, Internet telephony (Voice over IP [VoIP]) and other multimedia applications shift from analogue and dedicated lines to using the Internet as an access technology. This can become a security problem because current attack mitigation techniques are often based on "a model of isolation, physically separating voice and data or using virtual LANs or VPNs" [1].

However, these new opportunities also bring new kinds of attacks and frauds that were unknown to traditional analogue telephony. Therefore it is important that measures and controls are in place to protect both customers as well as the Internet and telephony service provider from malicious hackers and financially-driven attackers. Internet security reports show that VoIP will become a more interesting target to criminals and malicious hackers: "Cyber criminals will be drawn to the VoIP medium to engage in voice fraud, data theft and other scams—similar to the problems email has experienced" [2]. VoIP security surveys show that flooding and denial of service (DoS) attacks are the main threats

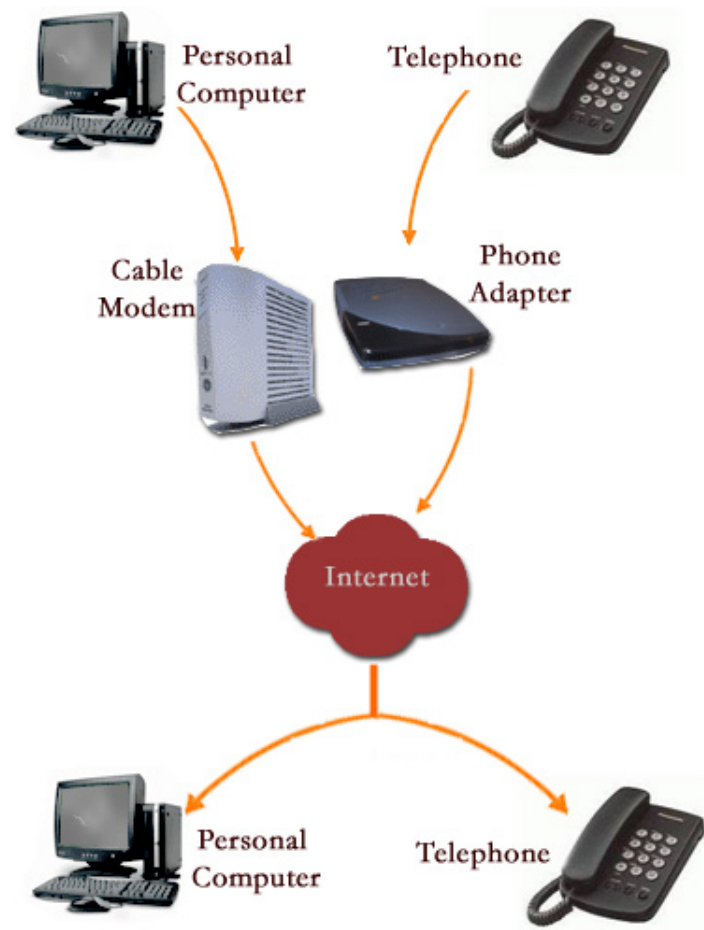


Figure 1.1.: Concept of VoIP, source: fcc.gov

to telephony systems [3, 4]. However, manipulation of sessions such as hijacking as well as passive attacks like eavesdropping can have a very strong impact on a person's privacy or a company's trade secrets. These threats are of great importance right after DoS attacks (figure 1, [5]). Recent publications of blacklists and attack patterns in the VoIP domain provide specific details of current threats as well as which hosts and network blocks are affected [6]. An example of a VoIP sensor running on UDP port 5060 by the Internet Storm Center is given in figure 1.2. It shows the distribution of attacks per day over the period of one month in 2010.

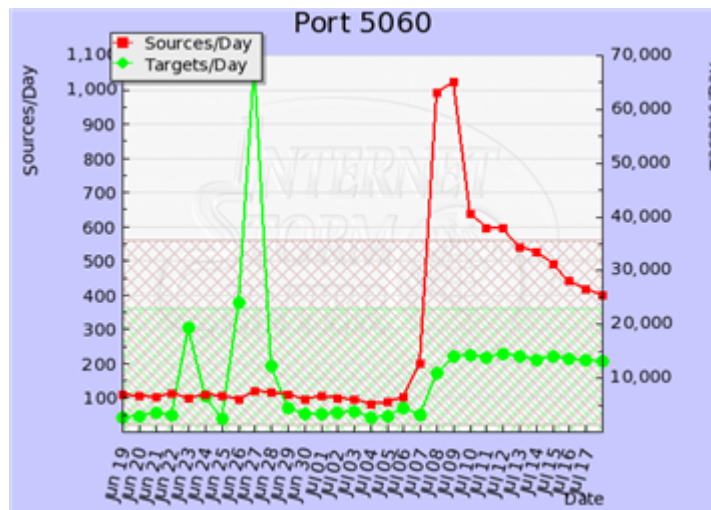


Figure 1.2.: VoIP scans, source: <http://isc.sans.edu/diary.html?storyid=9193>

1.1.1. Security Architectures

Traditional security architectures often consist of several firewalls (defense in depth [chapter 3 [7]]) and other security mechanisms to prevent attackers from eavesdropping on or penetrating the network (Figure 1-3 [8]). However, traditional firewalls are quite limited in what they can detect and filter (chapter 7 [9]). Application level firewalls exist that are able to detect attacks from the outside or information leaks from the inside. Snort is one example of an application that can detect patterns in application level network data [10].

Each network that is used by a large number of users has weak spots that cannot be mitigated with firewalls alone. Classification and detailed descriptions of attacks are given later in this thesis. Due to the number of network devices and because of the way humans use computers, there will be an insecure password, a faulty security setting, or a vulnerable application somewhere on the network [11]. In the end, an attacker only needs one entry point to gain access to the whole network.

1.1.2. VoIP

Technologies for securing the VoIP connection from one endpoint to the other are often not feasible because a network of trust has to be established before the communication takes place. Also, certain parts of SIP messages have to remain unencrypted (from an end-to-end perspective, they can always be encrypted between single hops) to enable SIP proxies to relay and redirect them. A hop-to-hop encryption can be established using the SSL/TLS protocol [12]. However, it has been shown that man-in-the-middle attacks can be conducted against SSL connections [13].

There are several reasons why simple and obvious solutions often do not work on large networks with many individual users. Such solutions are the use of application layer protocol-independent end-to-end encryption or secure protocols such as SRTP. They might fail because of several reasons:

1. Insufficient processing power at end-devices (hardware phones),
2. missing trust model for certificates and keys, or
3. missing incompatibility because of different protocol implementations.

Also, recent developments in consumer software (particularly the Windows operating system) have shown that a large number of logs or warning messages are not necessarily helpful. In fact, they often cause more harm by making the user immune to warnings. A very popular example is SSL certificate errors in web browsers that most users ignore after a while. It is safe to say that good security should happen in the background with as little noise and user interaction as possible.

Therefore, we propose a novel security architecture that uses well-known network and computer security methods to detect an attack at one part of the network and improve the protection for the whole network. The architecture only detects and mitigates a certain class of attacks which is described later.

1.2. Network and Information Security

Network and information security is defined by four key requirements that have to be met at all times to guarantee that the businesses and needs of companies, service providers and end-users are not disrupted by attacks, accidents, natural disasters, or random alteration of data due to noise on communication channels. These four requirements of security are (chapter 21.1 [14]):

1. Confidentiality: data must only be accessible to authorised parties
2. Integrity: data must not be modified by unauthorised parties
3. Availability: data must be available at all times
4. Authenticity: ability to verify the identity of a user

The increasing number of powerful and multifunctional mobile devices such as tablet PCs and smartphones provide new challenges to network security. Examples for such devices are the Apple iPad and phones with the Android operating system, respectively. The threat to network and information security comes from two main aspects: larger user base and limited security features.

The user base is becoming larger with the rise of mobile devices because more people are able to connect to the Internet without needing knowledge about how to buy, connect, and configure a traditional personal computer. Many of those users want to use emerging services such as VoIP wherever they go. Mobile broadband Internet connections enable people to be online even if they are not within range of a WLAN access point.

Due to the manufacturers' goal to build smaller and lighter devices that come with many specialised hardware components (GPS, FM radio), the processing power and variety in input methods on those devices are very limited. This is especially true for small smartphones that are most likely to be used in VoIP scenarios. Also, limited battery life makes it less feasible to run traditional antivirus solutions since they rely on regular or constant scanning of files, internet connections, and running processes [2].

However, there is a positive aspect about mobile devices regarding protection against network-based attacks. Most modern mobile operating systems are shipped with very strict rules that restrict the damage a single compromised program can cause. Android by Google is one of the most popular mobile operating systems. The operating system separates programs by assigning each with its own unique user and group ID. As a result, a malicious program usually cannot read or write data that belongs to another (legitimate) program. However, programs often need extended permissions to share and modify common data. The user has to explicitly give the program permission to do so. This is often done without much thought towards network security and Internet threats [15, 16].

1.3. Scope

The research and implementations for this thesis concentrate on protecting the customers and users of the VoIP network. Parts of the infrastructure that are excluded from the security architecture are:

1. The protection of VoIP backend servers, that is proxies and registrars, however they might be used for collection of data,
2. any computers and devices that are not within the protected network,
3. physical security considerations, that is physical access control or TEMPEST, and
4. individual protection mechanisms such as anti-virus products.

1.4. Outline

The following chapters of this thesis are as follows:

chapter 2 introduces the terminology and basic knowledge necessary to understand how VoIP works on both the client as well as the service provider side,

chapter 3 describes common attacks on VoIP and countermeasures for single attacks,

chapter 4 puts these previously existing and basic technologies and methods into the context of this thesis,

chapter 5 describes existing tools used to implement the testbed and the novel security architecture,

chapter 6 describes the novel security architecture developed for this thesis,

chapter 7 explains how the novel architecture achieves its goals on a technical level,

chapter 8 describes the evaluation process and the results in terms of performance and improvement of security,

chapter 9 concludes the thesis and gives an overview of future work in this area.

2. Voice over IP (VoIP)

VoIP follows the principle of end-to-end system design by pushing the intelligence of the network to the edges, thus taking load off the VoIP servers in the core:

The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible.

– chapter 1 [17]

This has advantages in performance and availability but it can also introduce problems, particularly regarding the security of the system. Following are a few examples of scenarios where the security is decreased by not having an intelligent network core. All of them assume that no end-to-end security measures like encryption on the IP layer (for example using IPsec) are in place.

1. Changing the route of packets: the end systems might only be interested in delivering the packets to the other client. Therefore, an attacker could perform a man-in-the-middle attack and eavesdrop on or modify the data.
2. Malware attacks the end-user's system: if the user's system or entire network is compromised, anti-virus and firewall software might be useless. This is particularly true for rootkit infections. In this case, a more intelligent network core might provide more security to the network user.

VoIP sessions usually involve several different protocol categories that are all necessary to transport multimedia data from one end of the conversation to the other as shown in figure 2.1.

1. Utility protocols: low-level protocols that are not specific to VoIP, for example ARP, IP, DHCP, or DNS.
2. Signaling protocols: describe and initiate VoIP sessions, thus helping to establish an end-to-end media channel. This thesis focuses on the signaling protocol SIP.
3. Media protocols: carry the actual data of the conversation such as audio (voice) and video. An examples of such a protocol is RTP.
4. Support protocols: needed for session and multimedia maintenance. SDP, RTCP, STUN, and NTP are the most common protocols.

Call Processing Signaling Protocols SIP	Voice	Support Protocols RTCP, NTP, SDP	
	RTP		
TCP or UDP	UDP	TCP or UDP	
IP			Utility Protocols ARP, DHCP, DNS
Data Link Layer			
Physical Layer			

Figure 2.1.: VoIP network protocol stack

It is also important to understand how SIP, RTP and their helper protocols work together in a VoIP session. This parallel configuration of SIP and RTP and the embedded nature of SDP are shown in figure 2.2.

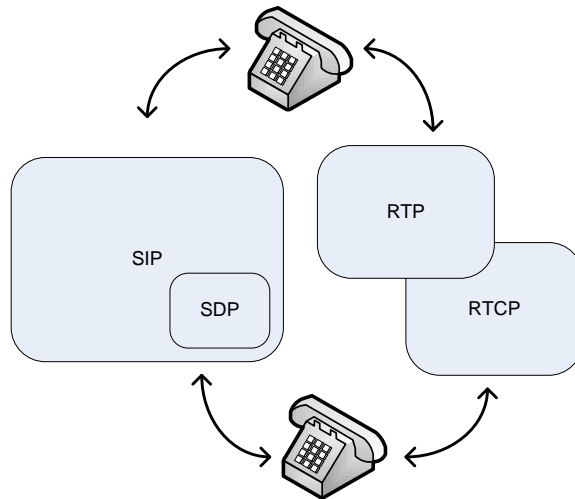


Figure 2.2.: Role of SIP and RTP protocols

2.1. History

Some of the protocols that are used within a SIP-based VoIP scenario are RTP [18], SDP [19], and SAP [20]. These existed in different environments such as the Internet multimedia conferencing architecture before SIP in its current version was developed and widely adopted (chapter 4 [21]). The first predecessors of SIP were packet-switched voice transmissions in 1974 and multimedia conference systems in the 1990s. One specific predecessor is the Multimedia Conferencing System [22]. It could be used for “point-to-point and multipoint teleconferences, with audio, video, and whiteboard tools.” Its Connection Control Protocol utilises UDP to establish connections between multimedia users.

The first version of SIP was called Session Invitation Protocol (as opposed to the current name of Session Initiation Protocol) or SIPv1. The first IETF draft for this new multimedia protocol was submitted in 1996. As mentioned earlier, supporting protocols like SDP and the protocols used in the network stack below the application layer, such as IP and UDP, existed already. An alternative protocol, the Simple Conference Invitation Protocol (SCIP), was proposed as an IETF draft in the same year [23]. SCIP made use of the exist-

ing email infrastructure: it used email addresses as unique identifiers and MX records in DNS.

The current version of SIP, version 2, is a combination of SCIP and SIPv1. Both protocols were merged at the 35th IETF meeting and the new protocol was named Session Initiation Protocol. SIP is still considered an unfinished draft. However, it is widely adopted as the large number of devices and software implementing the current specification of SIP and some of its extensions proves. Since 1999, SIP has its own working group in the IETF. The group was split into two parts later on (2001): one for the main specification and extensions, while the other is responsible for discussions about specific applications.

2.2. Components

2.2.1. User Agents

VoIP end-user devices such as telephones or computers are called User Agents (UAs). A software application that is used as a VoIP telephone on a computer without the use of dedicated hardware is called a softphone. UAs always consist of a User Agent Server (UAS) and a User Agent Client (UAC). Server and client receive and send requests and responses, respectively.

One problem with softphones is that they have to be on the same VLAN as the computers they run on. These computers also run and use many other potentially vulnerable network services, for instance sending and retrieving information to/from HTTP servers. Therefore, it is no longer possible to separate voice and data traffic. If an attacker gets access to a data device (for instance through phishing, malware, weak passwords, etc.) he can instantly attack the communication channels of a network as well.

A selection of popular hardware VoIP phones are shown in figure 2.3. Figure 2.4 gives examples of some softphones.



(a) Cisco phone

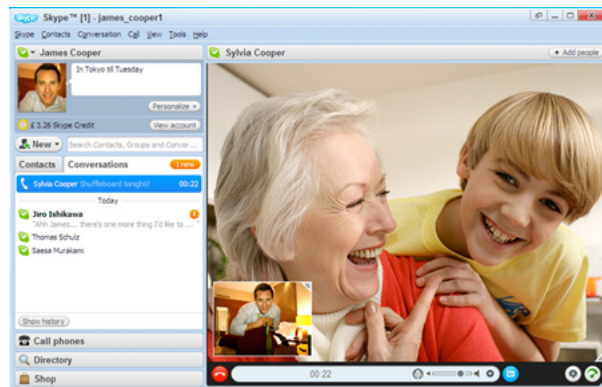


(b) Snom 820 phone

Figure 2.3.: Hardware phones



(a) X-Lite [24]



(b) Skype [25]

Figure 2.4.: Softphones

2.2.2. Servers

Different kinds of VoIP servers are necessary to make the UAs work together. Similar to an email address, the SIP VoIP address makes it possible to reach a person independently from his current location. However, redirect and proxy servers are necessary to tell the caller how to reach this person, that is tell him the IP address of the callee. Often, several functions that are described as individual servers in the paragraphs below are combined in one physical server.

Registrar

When a user signs in to his account he has to send a SIP REGISTER request to a registrar server. This is also where user authentication takes place to prevent unauthorised access to the telephony service. The registrar server then stores this information on a location server to make it available to everyone who tries to reach a certain user on this domain.

Location Server

A location server is a database where locations of VoIP users are stored. This is necessary to redirect callers to the correct location of the callee. For example, Alice's registered SIP URL could be `sip:alice@company.example` but the actual location of Alice is `sip:ali01@university.ac.nz` between 9 am and 5 pm.

Private Branch Exchange (PBX)

A PBX switches telephone calls within a domain, for instance within the internal VoIP network of a company. The lines that connect a PBX with another network, such as the public switched telephone network (PSTN), are called trunk lines. Usually, PBXs also act as registrar servers. Therefore, users have to authenticate against a cryptographic

challenge sent from the PBX/registrar in order to be available for incoming calls, and to be able to place calls or other types of SIP requests.

Proxy and Redirect Server

These two kinds of servers do the same thing from a user's perspective: they redirect a call from the server the callee is registered on (the host part of the SIP URL) to his actual location. However, proxy and redirect servers do that in different ways. While a redirect server *tells* the caller the current location or the address of the next redirect/proxy server, a proxy server transmits all VoIP traffic *through* itself on behalf of the caller's UA.

2.2.3. Session Border Controller (SBC)

Routers only inspect traffic at TCP/IP level 3 because all the information they need to route data can be found in the IP packets. Firewalls that protect the boundaries of a network only inspect packets at TCP/IP layers 3 and 4 (chapter 2.3 [14]). Therefore, a mechanism is needed that provides security and services at the application layer (TCP/IP layer 5).

SBCs are multimedia firewalls that work on the application layer. For the telecommunication provider an SBC is essential to protect the infrastructure that works in the background by controlling the signals and media streams that try to enter the core of the network. It also controls call admission at the border of a network. Either all (single-box setup) or part (dual-box setup) of the SBC sits in a Demilitarised Zone (DMZ) between the public network and the service provider's internal network [26].

One of the functions of an SBC is to operate as a SIP back-to-back (B2B) UA, that is it receives VoIP calls from one side of the border and establish a new connection to the other side of it.

Since firewalls are not aware of SIP sessions (traditional firewalls only examine traffic on layer 4 and below), SBCs provide mechanisms to establish SIP sessions from end to end through firewalls that perform Network Address Translation (NAT).

2.3. Protocols

2.3.1. Session Initiation Protocol (SIP)

SIP is used to initiate and control VoIP sessions. It uses TCP or UDP on top of the IP protocol and is located in the application layer of the ISO/OSI network model (chapter 2.3 [14]). SIP was designed as an end-to-end protocol, which means that the intelligence is located in the end systems and not in the core network (page 109 [21]). To exchange information about the multimedia session, for example which video and audio codecs to use, the Session Description Protocol (SDP) is embedded inside SIP messages. Therefore, by embedding other protocols like SDP the functionality of a packet goes beyond what SIP can provide alone. These basic stand-alone features of SIP are:

1. Registration and user location
2. Adding or removing participants from a session
3. Feature negotiation for a new session
4. Changing features during a session

The caller sends an INVITE message to the callee to initiate a multimedia session, for example a VoIP call. The callee *may* answer with a “180 Ringing” message (provisional) and *must* answer with a “200 OK” or error message. If there is no answer from the callee, the INVITE request will eventually time out. In order to tell the other party the specifications of the multimedia stream that will carry the actual voice signals, the caller has to embed an SDP message inside the SIP message’s body. He will also get an SDP message with parameters for the RTP stream from the callee [27].

After the successful start of a session the media such as VoIP audio is sent using the Real-time Transport Protocol. The messages transmitted by each party in a typical VoIP session using SIP are shown in figure 2.5.

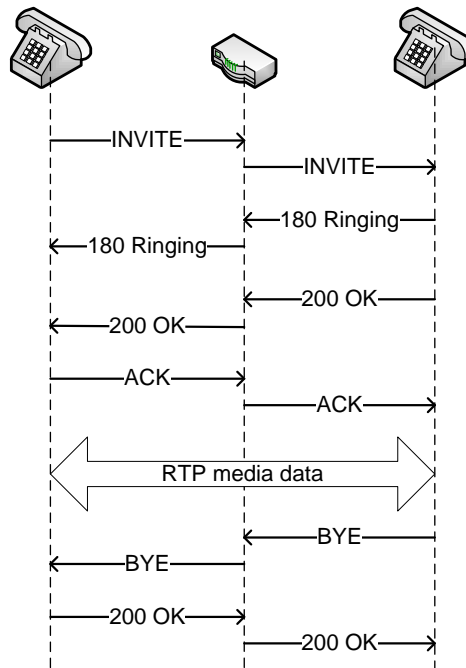


Figure 2.5.: VoIP conversation using SIP. The VoIP party in the middle is a SIP proxy that relays SIP messages to help to locate a user. RTP media data is usually sent directly between the end-points.

Table 2.1 shows a list of the most important, and often mandatory, SIP headers [21].

Header	Description
Via	Route of the packet
From	Public address of sender
To	Public address of receiver
Contact	Contact information of sender (optional)
Call-ID	Random ID of call session
CSeq	Request sequence number
Allow	Supported SIP requests (response to OPTIONS)
Max-Forwards	Maximum number of hops
Content-Type	Type of body (e.g. SDP)
Content-Length	Length of body in bytes
User-Agent	Phone identifier (optional)

Table 2.1.: Important SIP headers

Some headers contain a random string that is used for identification purposes. Even though these additional random parameters are essential parts of the SIP specification, they are only discussed briefly because they are generated and processed by the existing VoIP software used in this thesis. They do not play an important role in the concept of the proposed security architecture to be described in 6.

The additional random parameters are used to uniquely and globally identify call relationships. They are also important for detecting request loops in a network. For instance, the `From` header is of the form

```
From: NAME <sip:EXTENSION@SERVER>;tag=RANDOM
```

with `NAME` being the full name or an alias of the person calling, `EXTENSION` being either the extension number or the nickname the caller is registered under on the VoIP server (given by `SERVER`), and `RANDOM` being a random alphanumeric string set by the calling phone. The `To` header can also contain such a random tag. The SIP protocol specifies that the tag is only to be used in peer-to-peer dialogs, that is SIP requests and according responses.

Each request must contain one or more `Via` headers which must have a branch parameter appended to the address of the routing node. Therefore, a `Via` header is of the form

```
Via: SIP/2.0/UDP ADDRESS;branch=RANDOM
```

with `ADDRESS` being the network address of the node that forwarded and routed the request (including the original sender) and `RANDOM` being a random alphanumeric string that “MUST always begin with the characters z9hG4bK” (section 8.1.1.7 [27]).

2.3.2. Session Description Protocol (SDP)

The Session Description Protocol is used to describe the parameters of a SIP session. Its most important feature is to tell the other party or parties the specifications of the

RTP multimedia stream, particularly the port number. Without this information, it is not possible to establish the RTP stream since, unlike the SIP server's port (default: 5060), the port number is different for each session (chapter 3 [21]).

SDP messages always start with a `v=0` line. Then, the sender of the message can specify owner, subject, description, and timing parameters. The meaning of several common SDP header lines is given in table 2.2 [19].

Header	Description
<code>v=0</code>	Protocol version
<code>o=</code>	Session owner
<code>s=</code>	Session subject
<code>i=</code>	Session description
<code>t=</code>	Timing information
<code>m=</code>	Media information
<code>a=</code>	Additional information

Table 2.2.: Important SDP headers

The most important line is the one containing the multimedia type, protocol, and port number, such as an RTP audio stream on port 20010:

```
m=audio 20010 RTP/AVP 0
```

With this information, the receiver of the message is able to connect to the sender's RTP stream on port 20010. After he has sent his RTP port number to the caller, both parties can send and receive audio signals, and therefore have established a full-duplex, bidirectional phone call.

2.3.3. Real-time Transport Protocol (RTP)

Multimedia content has to arrive at the receiver shortly after it has been sent to make sure that the inevitable delays do not make conversation impossible. Even more important is the correct order of packets. Therefore, each RTP packet contains a timestamp in its

header to make sure the packets are replayed in the same order as the sender transmitted them. The timestamp is also used to drop packets that are too old.

RTP only provides a way to transport multimedia data. The data itself has to be provided in a way that is suitable to send it over a network, that means it has to be digitised and preferably compressed. This is achieved by utilising codecs that are appropriate for the job. An example of a voice audio codec is G.711 (chapter 1 [21]).

To synchronise audio and video streams the Real-time Transport Control Protocol (RTCP) is used [18]. It always uses the User Datagram Protocol (UDP) as the transport layer protocol (Figure 3-5, [21]). RTP multimedia streams are often unencrypted but an alternative protocol – the Secure Real-time Transport Protocol (SRTP) [28] – exists.

2.3.4. Session Traversal Utilities for NAT (STUN)

Network address translation (NAT) is a technique that is used for security (obscuring the internals of a network), abstraction (not being dependant on IP addresses provided by an external provider), and most importantly, because the number of IPv4 addresses is very limited (both in general and per customer) [29]. However, many services have problems running behind a router that uses NAT. VoIP using SIP is one of those services because it uses random ports that are usually not accessible from the Internet (no port forwarding).

The STUN protocol provides a way for VoIP devices to be reachable behind a NAT router [30] by making the UA connect to a dedicated STUN server. STUN uses UDP as the transport protocol, and therefore has to provide its own mechanisms to guarantee reliable delivery if necessary.

The basic operation of STUN consists of a request-response transaction using the “Binding” method. Because a STUN request passing through one or more routers performing NAT has its source IP address – and possibly also the port – changed, the STUN server is able to record the public IP address and port of the client sending the request. Then, it sends this public information back to the client. This way, the client learns about its

public IP address and port, and can send them to other hosts it is connecting to. Note that restricted NAT only allows incoming connections to reach their targets after the internal client has sent a packet to the target host to open the channel.

2.4. Security

2.4.1. VoIP Applications

VoIP packets are routed using the common Internet Protocol. Therefore, they are part of the Internet traffic that connects millions of computers worldwide. For malicious hackers it is easier to intercept or reroute VoIP packets than to tap a traditional phone line because they can do it remotely. Possible attack vectors at different parts of the VoIP infrastructure are given in figure 2.6. It shows two end-users connected to the rest of the VoIP network via a firewall server on the service provider's side. This server represents the protective measures of the provider. However, in reality, several servers and firewall appliances are likely to be used. The end-users' phones are connected to the Internet via residential gateways (RGWs) that act as a router and VoIP gateway.

Because of these security concerns, VoIP traffic should always be encrypted. One of the most commonly used applications, Skype, uses a proprietary protocol to accomplish secure communication [25]. However, due to its obfuscated nature, Skype is not the primary choice for many companies.

Possible attacks and ways to misuse VoIP are: [31]

1. Spit (Spam over Internet telephony)
2. Eavesdropping
3. Denial of service (DoS) and flooding
4. Toll fraud

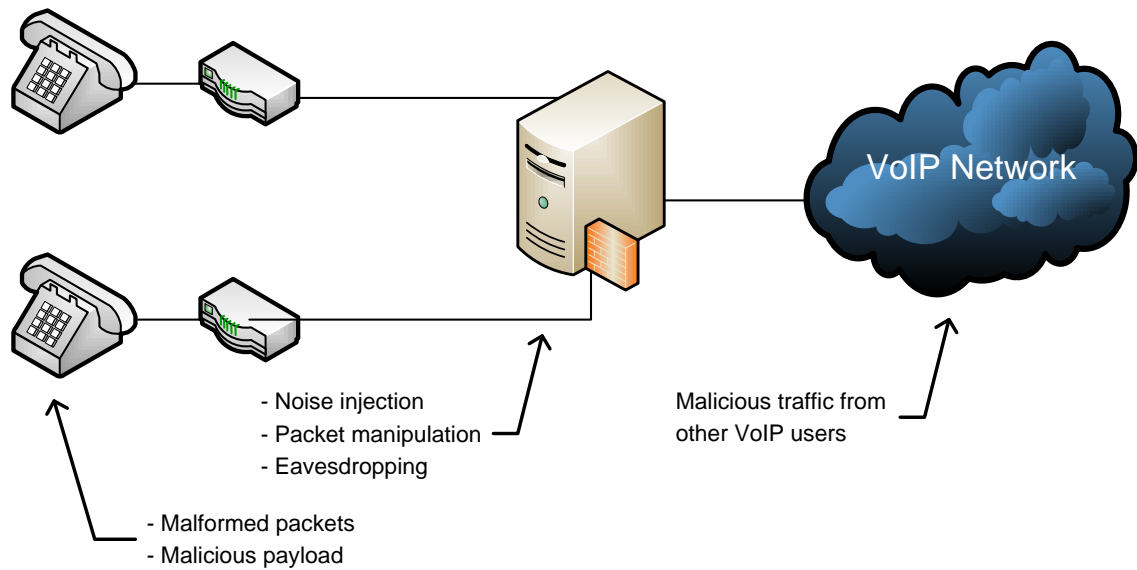


Figure 2.6.: VoIP attack vectors with end-users' phones, RGWs, and service provider's backend.

5. Noise injection and malicious traffic

While many approaches to the threat of flooding and DoS attacks have been proposed [32, 33, 34], other threats are still a major problem to both consumers as well as service providers. Often, security architecture approaches have to be taken to provide a holistic protection technique against sophisticated attacks.

2.4.2. VoIP Networks

VoIP networks are difficult to keep secure because much of the network intelligence is pushed to the edges of the network, that is to the UAs. The VoIP servers only redirect sessions and help users to find each other. Often, an established VoIP session does not even have to use these servers. Instead, end-users can communicate directly with each other.

SIP provides authentication, integrity, and confidentiality by relying on existing security mechanisms (chapter 6 [21] and section 22 [27]). One of these mechanisms is Secure

Multipurpose Internet Mail Extensions (S/MIME) which provides public-key encryption and digital signatures [35].

To authenticate a VoIP user, that is to make sure that the SIP message comes from the user who is given in it, the server sends back a challenge. The user has to respond correctly to this challenge in order to prove his identity. An attacker is still able to change parts of the message without any of the UAs involved noticing. The only part he cannot simply change is the challenge response because it uses a cryptographic hash function. That means that an attacker who is able to intercept an authenticated message can change the remaining fields of the SIP request or response in his favour. In order to prevent attacks on the remaining parts of the message, further measures are necessary.

Integrity and confidentiality are both achieved using S/MIME. A user can sign a message with his private key so that other users can verify the integrity of the message with the sender's public key. In a very similar way, a user can encrypt a message to other users by using their public keys (chapter 21.4 [14]).

It is clear that not all headers of a SIP message can be end-to-end encrypted since a proxy server on the route from one UA to the other may have to analyse and modify certain header lines, particularly the Via lines.

When looking at the security between an end-user and an ISP, connections are relatively safe from wiretapping due to the widespread use of fiber optics instead of copper wires (chapter 2 [36]).

Because of the small number of different SIP and SDP messages, normal (anomaly-free) VoIP network traffic can be easier to predict than legitimate generic network traffic (a mix of different applications). Therefore, it is easier to detect anomalies and malicious behaviour in isolated VoIP networks.

3. Attacks and Countermeasures

This chapter describes several techniques to exploit flaws in the SIP protocol, one of the other protocols used in a typical VoIP session, the VoIP server software, or the UA software. Additionally, mitigation techniques against these flaws or the resulting attacks are described. The purpose of this section is to give an overview of what is possible in a SIP-based VoIP environment from the attacker's perspective as well as from the network administrator's and security expert's point of view.

A look at previous and related work that surveyed or monitored the security of production VoIP systems shows that threats like tool fraud, identity theft, denial of service attacks, and SPIT calls are a problem for most service providers [1, 6].

3.1. Introduction

Table 3.1 categorises common attacks and threats into a social and a technical category. Only those attacks and threats that are likely to occur in a VoIP scenario are listed. Specific attack scenarios are described in the following sections.

3.2. Social Threats

As with any large interconnected community (user base), it is relatively easy for an attacker to get information about users in an anonymous way. The collected information

Social	Technical
Identity theft	Flooding attack
Credential theft through phishing	Credential theft through eavesdropping
Rerouting	Rogue servers (Man in the Middle)
—	Software crashes (DoS)
—	Network congestion (DoS)
—	Brute force (passwords)

Table 3.1.: VoIP attack methods

makes it possible to target a specific user in a way that does not appear to be an attack. The technique of attacking someone using social skills and additional information about a specific target is called social engineering.

Social threats are dangerous not only to VoIP users but threaten all Internet users, in particular those with accounts on social networking websites and other online services. The following sections describe various possible social attacks against VoIP users and their consequences.

3.2.1. VoIP Phishing (Vishing)

As with all new technology, it does not take long until criminals discover the new services and try to turn them into money. Even though Phishing is not an attack on computer systems itself, it nevertheless gives the attacker access to it by using social engineering techniques on legitimate users of the system. Often, a user is tricked into believing that a legitimate party, such as his VoIP service provider, needs information about his account to enable or restore certain functionality. If the user enters his credentials into a fake login form or sends it directly via email, the attacker can collect this information and use it to log on as the specific user.

The collected information can then be used to conduct fraudulent activities in the name of the victim. If money services are directly connected to the user's account, it is easy for the attacker to achieve a financial gain. A common scenario in VoIP environments is to make a user call premium numbers which usually cost several dollars per minute. The

premium numbers are operated by the criminals and provide financial income with each call.

3.2.2. Rerouting and Man-in-the-Middle Attacks

If a user can be tricked into configuring his phone so that it routes packets through the attackers network space, all or parts of his communication are revealed and potentially compromised, depending on the security of the network (authentication only or integrity checks and encryption). In a process similar to the previous section (Vishing), an attacker could send emails that resemble service emails from the VoIP service provider to end-users. These emails could include directions to change the address of a VoIP proxy server. Users that follow these fake instructions will fall victim to man-in-the-middle (MITM) attacks.

3.3. Technical Threats

This section describes technical threats that can have a serious effect on users and networks even if they are protected against the attacks mentioned in the previous section. These attacks often use a lot of resources to achieve sending malicious data with high frequency.

3.3.1. Denial of Service

A Denial of Service (DoS) attack renders the target system useless. Possible outcomes of DoS attacks are denial of access to information, applications, computer systems, or communication channels (chapter 2 [8]). This can be achieved in many ways, for example flooding the VoIP network with SIP messages or changing the registration of VoIP phones

on the registrar server. The latter can also be used to redirect VoIP traffic to non-legitimate users (for specific attacks see chapter 12 and 13 [37]).

An important subsection of DoS attacks are Distributed Denial of Service (DDoS) network attacks (chapter 7 [38]). In principle, DDoS attacks work exactly like DoS attacks but utilise more than one single source to send malicious packets over the network to the victims' computer systems. Often, attackers use other computers without permission by using covert malware, also called trojan horses or rootkits (see below). These infected computers are called zombies or bots. A bot is controlled by a master (the original attacker) and all bots that belong to a master form a botnet.

3.3.2. Circumventing Password Authentication

Given a large enough user group, an attacker will almost always be able to find a weak password. A weak password could be either too short so that it can be cracked with a brute force attack, too easy to guess ('1234'), or a word that can be found in a password dictionary. Recent research and publications by password recovery companies have shown that many users use very simple and predictable passwords, often on many websites simultaneously [39, 40] (see figure 3.1).

Therefore, authenticating clients and servers before the communication starts is a very limited method to ensure that only legitimate end-points communicate with each other. It can only provide a basic first step towards a secure architecture.

3.3.3. Rogue SIP devices

Endler and Collier demonstrated several ways to insert a rogue SIP UA or server into a VoIP network (chapter 6 [37]). To establish these, VoIP UAs have to be tricked into registering to a rogue registrar or redirecting all traffic through a rogue proxy. Therefore, either a social engineering attack has to be done before the rogue server is set up, or a

However, modified binaries can be detected by booting the system into a different rescue system and scanning the memory and storage space of the infected system (Introduction [43]).

Guillaume Delugré recently demonstrated that a rootkit can be directly installed to a network card by modifying its firmware [44]. A similar approach of reverse engineering is also possible for VoIP hardware phones. This is particularly true because more and more devices run a Unix-based embedded operating system that is similar to common desktop and server systems. Therefore, a user's telephone or computer that runs a softphone cannot be trusted even if the operating system, firmware, and (in the case of a personal computer) applications, antivirus and firewall software are up to date.

3.3.5. Fuzzing

Fuzzing is also known as robustness testing or functional protocol testing (chapter 11 [37]). The technique is used to find software and protocol vulnerabilities. Instead of sending predefined input to a service like a SIP server, traffic is generated according to rules. Commercial tools usually come with thousands of tests preloaded. For example, the Codenomicon SIP test tool [45] comes with 35,000 test cases including INVITE, OPTIONS, and REGISTER SIP messages.

3.3.6. Malicious Traffic in the SIP Message Body

The SIP protocol specification [27] only specifies the content of the SIP message line and the SIP headers. The message body on the other hand can contain any arbitrary data. This data can even be encrypted since the SIP message body is only used by the end points of a session. Besides the obvious example of SDP messages, there are other use cases. Since several message bodies can be contained in one SIP request, a user could send an image as well as an SDP message inside a SIP INVITE request. It is then up to the UAS to interpret both bodies (the SDP message as well as the image data).

Considering the many different VoIP clients and the amount of different features (like sending an image with an INVITE request), it is very likely that software bugs will enable a malicious hacker to crash the UA or introduce malicious software into the operating system. Since modern hardware phones are computers with an embedded operating system too, it does not make a difference if the target is a traditional computer with a softphone or a hardware phone.

4. Related Work

A lot of work has been done on VoIP security in the past. This chapter describes public research results on VoIP. An important subsection of this research is on network attacks and countermeasures.

4.1. VoIP Attack Surface

Recent Internet security threat reports and articles from security researchers [2, 46, 47] show that the VoIP attack surface is getting bigger. This is both due to more people using VoIP services as well as more malicious hackers and criminals focusing on attacking these new telephony users.

The Australian chapter of the HoneyNet Project deployed VoIP honeypot sensors (called phoneyNet) to look for scans and attacks in the Australian address space in 2009 [46]. VoIP scans only occurred every few weeks, originating from all over the world and were far less frequent than traditional computer virus and DDoS attacks in the same address space.

This changed in 2010 as VoIP scans became more and more common [48]. [47] describes detailed results of 2083 VoIP events collected over the period of nine months on the deployed ADSL-based honeypot. The results of the analysis show that most malicious VoIP traffic comes from China (over 63%). This does not necessarily indicate that the location of the attackers is in China but rather that there are many compromised computer

systems in China that are used to attack VoIP systems worldwide. The second largest group of attacks originated from an unknown location which “was composed of either IP numbers that did not result in a resolution in the geographical IP databases as it was unknown or was a spoofed unassigned IP number.”. The scanning hosts were identified as Asterisk-based (see chapter 5), SIPVicious, and sundayddr (a modified SIPVicious scanner). The similarity between the SIP messages used in these new scans that emerged in the second half of 2010 and the ones created by existing VoIP security tools was also noted by the developer of SIPVicious. He explains that one of the techniques used to bypass network security measures is “distributing the scans across different IP addresses” [49]. This shows the need for an event correlation mechanism in a VoIP network which is central to the proposed security architecture to be described in chapter 6.

4.2. Protocol Improvements

Several protocols exist that are aimed at making VoIP communication secure. The Secure Real-time Transport Protocol (SRTP) uses modern encryption standards to provide “confidentiality, message authentication, and replay protection to the RTP traffic and to the control traffic for RTP” [28]. Another example of an improved VoIP data protocol, ZRTP, was developed by the creator of the data encryption program PGP [50]. However, in the case of both protocols, all endpoints have to support the additional or enhanced protocol version. This cannot be guaranteed, in particular for older hardware phones.

4.3. Attack Countermeasures

4.3.1. Denial of Service and Flooding Attacks

Lee and Hunt propose “a novel method to address the protection necessary to mitigate flooding attacks in VoIP networks” by extending the SIP authentication method and in-

roducing a firewall nonce checking mechanism [32]. The firewall creates a nonce and sends it back to the calling UA. The UA then resends its INVITE request including the hashed response which is calculated using the client's credentials and the nonce from the firewall. The firewall does not check the response but only the nonce. The response is simply forwarded to the SIP server that the client originally attempted to connect to. The validity check of the nonce in the firewall uses essential SIP headers that are critical to security. These headers are To, From, Via, Call-ID and CSeq. This way, if the attacker tries to change the values of these fields, the nonce becomes invalid. Because the authentication is request by the firewall in a stateless manner, the attacker cannot exhaust the VoIP server.

VoIP IDS can detect and block flooding attacks, for example by using finite state machines (FSM) to detect malicious SIP requests as proposed by Sengar et al. [51]. This way, unusual and potentially malicious SIP messages can be detected at the IDS and filtered before they reach any important SIP equipment. The authors of the paper propose a protocol-specific IDS (called vIDS) "to detect any deviation from normal system behaviors, and hence, capture unknown attacks."

4.3.2. Eavesdropping

Apart from the VoIP protocol improvements mentioned above, mechanisms that protect the users confidentiality can be installed in other layers of the network stack, mainly in the IP protocol. Virtual private networks (VPNs) can provide a secure tunnel between two end-points. One example of a more secure IP variant is IPsec [52] and another (open-source) VPN implementation is OpenVPN [53].

4.3.3. End-user Protection

Charney [54] proposes a large-scale collective defense mechanism against computer malware by adopting known public health models. He points out that such a quarantine

technique can only work if malware can be detected easily and reliably. While many enterprises and government organisations already have the staff and technical capabilities to realise quarantines for computer systems, most Internet users are not protected by such malware countermeasures.

As Schneier [55] points out, Internet quarantines pose a risk to unrestricted and unlimited access to the Internet. Missing software patches, insecure operating systems or applications that fail to provide a valid certificate could become indicators that would put a computer into quarantine. It is clear that many privacy and legal concerns make the implementation of a quarantine system difficult.

4.4. Event Correlation and Anomaly Detection

Rieck et al. [33] describes a self-learning intrusion detection system for VoIP systems that can detect any SIP traffic, that is session setups and tear down messages, that deviate from previous training data.

Different approaches to detect whether a stream of data contains telephone voice have been given by Zissman [56]. Such a mechanism is needed to detect malicious RTP streams that pretend to contain audio. However, attacks that exploit vulnerabilities in RTP applications by sending content different from the expected audio are very rare.

Rules to detect some SIP attacks have been published for the open-source intrusion detection software Snort [57].

Distributed approaches to detect anomalous and malicious activity are an important and large field of ongoing research. The technique of offloading processing and analysis of potential malware to network services can be essential to devices with limited resources such as mobile platforms [58, 16]. Within this distributed security architecture, additional services such as honeypots have been implemented to help with classification of malware [59].

4.5. VoIP Network Simulation

The simulation framework OMNeT++ in combination with the extension INET (as described in chapter 5) provides all the mechanisms needed for a VoIP network except VoIP application protocols. VoIPTool is an extension for INET that provides modules that simulate realistic VoIP data transmission [60]. However, it does not contain any modules for the SIP protocol.

5. Tools

This chapter describes the tools that have been used inside the testbed to evaluate attacks against VoIP systems and their appropriate countermeasures. Whenever possible, open-source tools were used. This is for two main reasons: the first is cost-efficiency in an academic research environment. The second is that generally open-source programs are easier to deploy in heterogeneous networks because it is possible to adapt the software to the specific requirements of the environment.

The first two sections describe the tools used to set up the testbed and the network simulation for security and performance tests (see chapter 8 on evaluation and results). The software that is used inside this testbed to achieve the goals of the security architecture (see chapter 6) are described afterwards. A list of version or build numbers for the software used for this thesis is given towards the end of the chapter. The chapter ends with a discussion of unsuitable tools, that is software or hardware that is not used in the experiments conducted within the security architecture testbed or network simulation.

5.1. Testbed Setup

In the first step, the novel security architecture described in this thesis has to provide results from a proof-of-concept setup using only a very limited number of end devices. This is done to ensure that the implementation of the architecture and communication between its components works as designed.

5.1.1. VMware Workstation

Virtualisation has made the development and evaluation of many different types of software, from operating systems to graphical user interfaces and even malware, much easier. VMware Workstation allows a user to clone existing virtual systems (VMware images). Therefore, a developer can save a lot of time setting up identical or nearly identical operating systems compared to using physical computers. Additionally, the internal state of these images can be observed for malicious activities and reset to a healthy state if necessary.

5.1.2. Virtual Machines

Since most of the tools, clients, and servers mentioned below either run solely on Linux or are easiest to install and manage on a Unix-based system, a small variety of Linux distributions has been installed on virtual machines to implement the proof-of-concept architecture:

- Debian Linux 5.0.4: very lightweight distribution (used for the Internet gateway)
- Asterisk NOW: based on CentOS 5.3 (used for the Asterisk VoIP PBX)
- Backtrack 4 final: based on Debian 5.0, includes many network security tools such as nmap and Wireshark (used for all end-users and attacking clients)

5.2. Simulation

In order to evaluate the performance of the security architecture concept in a large VoIP network, a simulation framework is used. Additionally, a controlled simulation environment makes it possible to develop and deploy new features as well as to collect data from different sensors across the simulated network in a quick and controlled process.

5.2.1. OMNeT++

OMNeT++ is a discrete event simulation framework written in C++. It can simulate different types of networks including communication between computers. Other possible applications are on-chip and queueing networks [61]. Additional projects exist that extend OMNeT++ with protocols from the TCP/IP stack and realistic VoIP data functionality [62, 63]. Many of those extensions have been written and used at universities all over the world [64, 65, 66]. Therefore, it is safe to say that OMNeT++ is widely used within academic environments. However, [67] discovered significant differences between simulations and testbeds in wireless networks. This underlines the importance of the testbed as a first step towards the novel security architecture.

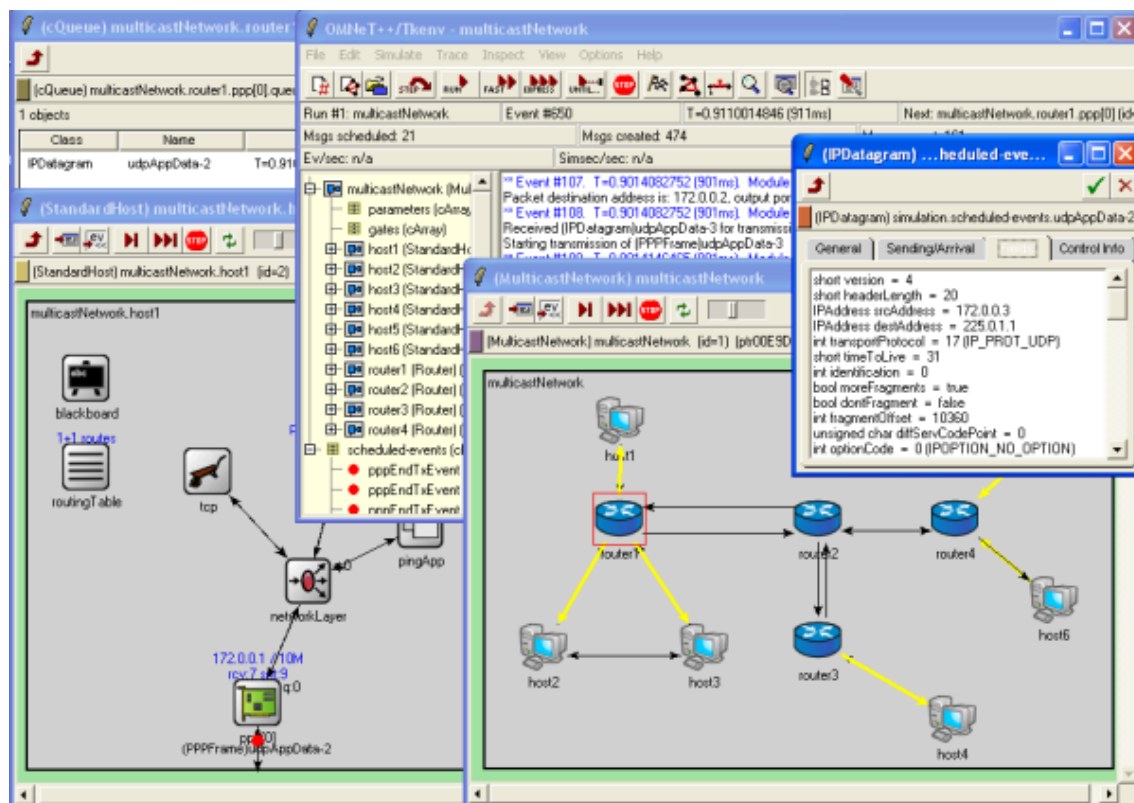


Figure 5.1.: OMNeT++ network simulation graphical user interface [61]

5.2.2. INET

INET is a framework for TCP/IP communication that is built on and integrates into OM-NeT++ [62]. It provides a set of simple and compound modules that resemble common devices and protocols found in a basic computer network environment. Examples of protocols which are realised as simple modules are Ethernet, IP, TCP and UDP. Compound modules that consist of protocol and utility modules are TCP/IP hosts, an Ethernet switch, an Ethernet hub, and routers (IPv4 and IPv6). Many other frameworks are built on top of INET to simulate peer-to-peer networks or mobile and ad-hoc environments. INET does not include any security-related modules such as firewalls or IDS. It also is not aware of application level protocols. Therefore, INET as it is provided by the open-source project cannot understand or act on VoIP traffic.

5.3. Traffic Generator

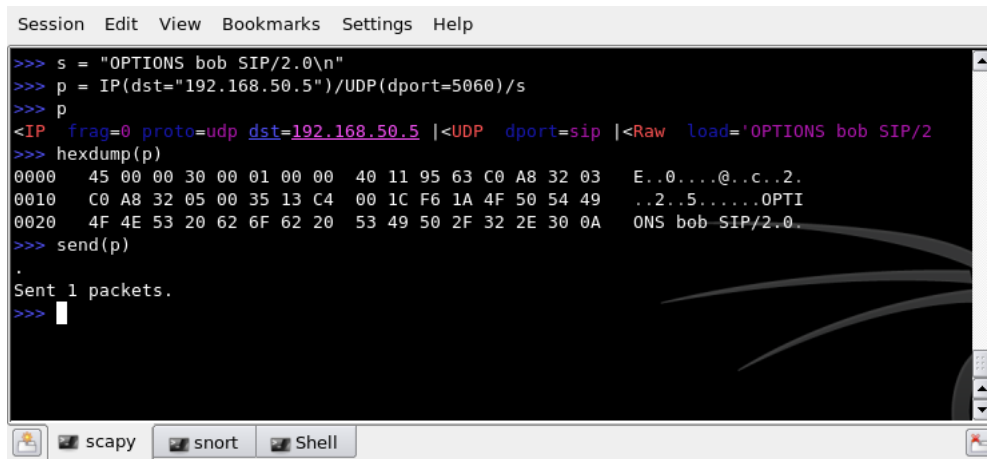
In order to test the performance and security of the security architecture, several methods and tools have been used to generate ordinary network traffic as well as specially crafted VoIP traffic. Scapy is a Python program for interactive network packet manipulation [68]. A trivial example of a Scapy command to generate and send an incomplete SIP packet as it might occur in an OPTIONS request scan is:

```
>>> s = "OPTIONS bob SIP/2.0\n"
>>> p = IP(dst="192.168.50.3")/UDP(dport=5060)/s
>>> send(p)
```

Scapy is not suitable for fast performance tests against a network. Since it is written in Python and uses several layers of abstraction, execution speed is slower and the memory usage is higher than that of other tools (chapter 6.3, [69]). The tool can either be used in an interactive mode by a network analyst or software developer, or its libraries can be

used inside Python programs. The latter method has been used to write scripts that send messages after a certain event in the event correlation engine has been triggered.

Figure 5.2 shows a screenshot of Scapy being used to generate a simple SIP OPTIONS request. All parameters of the SIP and underlying protocols can be spoofed.



```
Session Edit View Bookmarks Settings Help
>>> s = "OPTIONS bob SIP/2.0\n"
>>> p = IP(dst="192.168.50.5")/UDP(dport=5060)/s
>>> p
<IP frag=0 proto=udp dst=192.168.50.5 |<UDP dport=sip |<Raw load='OPTIONS bob SIP/2
>>> hexdump(p)
0000  45 00 00 30 00 01 00 00  40 11 95 63 C0 A8 32 03  E..0....@..c..2.
0010  C0 A8 32 05 00 35 13 C4  00 1C F6 1A 4F 50 54 49  ..2..5.....OPTI
0020  4F 4E 53 20 62 6F 62 20  53 49 50 2F 32 2E 30 0A  ONS bob-SIP/2.0.
>>> send(p)
.
Sent 1 packets.
>>>
```

Figure 5.2.: Scapy sending an incomplete SIP packet using the Backtrack 4 Linux distribution

5.4. Scanning and Enumeration

Basic tools such as ping, nmap (a port scanner), or captured network traffic (for example using Wireshark) are usually used to determine the state of a network in case of problems that may occur. However, in a VoIP environment, specialised tools that make use of the SIP protocol in the application layer can provide a more detailed look at the situation. For example, while a tool like ping is enough to determine if a host is up and running, a “VoIP ping” can query specific information from an active SIP device. This can become an important issue if SIP servers are configured to run on a different port than the default one (5060).

While, in principle, it is possible to generate all necessary diagnostic SIP messages using a traffic generator like Scapy, an easier and more reliable solution is to use existing VoIP security software.

The VoIP security suite SIPVicious by Sandro Gauci consists of several tools with different purposes [57]. The main tools are:

1. svmap: scans given IP addresses for the presence of a SIP server
2. svwar: identifies working extensions on a PBX
3. svcrack: tries to brute-force digest authentication passwords

5.5. VoIP Infrastructure

5.5.1. Asterisk

According to the website of the Linux distribution that is used in the security architecture testbed for the VoIP PBX, AsteriskNOW [70], Asterisk and its extensions can be used in many different scenarios. A few examples of applications are given below:

1. VoIP and Skype gateway
2. Voicemail system
3. Call recorder
4. Fax and speech server

A graphical user interface accessible through a web browser can be used to configure the parameters of Asterisk without the need to edit text configuration files, which is more error prone than a guided installation and configuration interface.

Through a manifold of add-ons or modules it is possible to extend Asterisk with many new features including basic SBC functionality.

5.5.2. SIP Phones

Closed and open-source softphones have reached different maturity levels. In terms of features offered to the user through the graphical user interface, the following ascending order among a selection of free softphones can be established:

1. Linphone: Open source, supports SIP
2. KPhone: Open source, supports multiple identities
3. Twinkle: Open source, multiple phone lines and automatic features such as call redirection
4. Skype: Closed source, proprietary protocol, supports calls to PSTN via external commercial servers and video chats

In this work, only KPhone and Twinkle are used because they are included in most modern Linux distributions and offer the functionality needed to conduct the experiments. Linphone is only used to evaluate the responses to a specific SIP request from different phones (see table 6.1) to obtain a more comprehensive picture.

5.6. Honeypots and Honeynets

In order to get statistics on the amount of VoIP traffic, both legitimate and malicious, a VoIP module for the low-interaction honeypot dionaea [71] was developed. Another VoIP honeypot with a different approach (it acts as a SIP phone and connects to a SIP registrar server that has to be present on the network) is Artemisa [72]. Dionaea was used instead of a dedicated honeypot software because it is already in use by many honeypot security researchers all over the world. Once the VoIP module was developed and integrated, it could be deployed worldwide with the rest of dionaea's source code.

Malicious hackers and operators of botnets also started using honeypots for their purposes [73, 74]. In this case, the botnet client software sets up a fake administration panel that is

accessible with a web browser and susceptible to SQL injection [75] and weak passwords (see chapter 3). It then presents random “malware statistics” to the authenticated user. The purpose of this mechanism is to provide security analysts and malware competitors with false information. Additionally, connections from security analysts and competitors can be logged and blocked in the future.

5.6.1. Dionaea

[The intention behind dionaea] is to trap malware exploiting vulnerabilities exposed by services offered to a network, the ultimate goal is gaining a copy of the malware.

– Dionaea website [71]

Dionaea exposes different services such as FTP (port 21), HTTP (port 80), SMB (port 445), and SIP (port 5060) to the network. It uses libemu to safely execute downloaded malware and record all system calls during the emulation process. Afterwards, a honeypot module can analyse the calls to determine the type and version of the malware. The honeypot modules for the protocols are written in the scripting language Python [76] in order to achieve flexibility and quick development cycles. The core components of dionaea are written in C. In order to call C functions directly from Python, a library wrapper is needed. Cython [77] is used to interface C data structures like the connection.

Dionaea offers a variety of ways to log the actions taken by an attacker: traditional text file logs, logging to an SQLite database which allows flexible and easy querying, and logging to a remote XMPP server. The latter is also used to exchange information and downloaded malware with other honeypots.

5.6.2. VoIP Module

Like the other modules, the VoIP honeypot module is written in Python. It exposes basic services of a SIP server such as responding to REGISTER, OPTIONS, INVITE, BYE, and CANCEL requests and establishing RTP voice multimedia sessions in response to INVITE requests. All the incoming and outgoing information of SIP requests and responses are logged to an SQLite database through dionaea's logging interface. Optionally, a complete binary dump of the RTP stream can be created as well. The relevant code sections are given in appendix E.

Since sophisticated and targeted attacks on VoIP users are not very common yet, the main goal of dionaea's VoIP module is to analyse the activity of scans and DoS attacks on large networks such as the Internet. Therefore, it does not operate on the incoming data, for example by analysing it for known malware signatures or by executing it in a safe sandbox. However, logged events of SIP scans alone can provide useful input data for a security architecture if they are correlated properly.

Figure 5.3 shows a UML class diagram of the VoIP honeypot module within dionaea. Some of dionaea's core classes such as the `connection` class that is used by all honeypot services are written in C and exposed as a Python class to the individual honeypot modules. `Sip` and `RtpUdpStream` both inherit from the `connection` class in order to provide services to the network. While there is only one instance of `Sip` which always listens on UDP port 5060, `RtpUdpStream` objects can exist many times. Each successful (authenticated) SIP INVITE request will cause the creation of a new `SipSession` and `RtpUdpStream` instance. The SIP session data structure is used to store parameters for each call individually, such as RTP port and authentication status. For most of the incoming SIP messages, the one instance of the `Sip` class merely forwards them to the correct `SipSession` instance. The key used to identify the correct session object is the Call-ID SIP header mandatory for each SIP message. A BYE or CANCEL request will delete the SIP session from memory if it has been created previously.

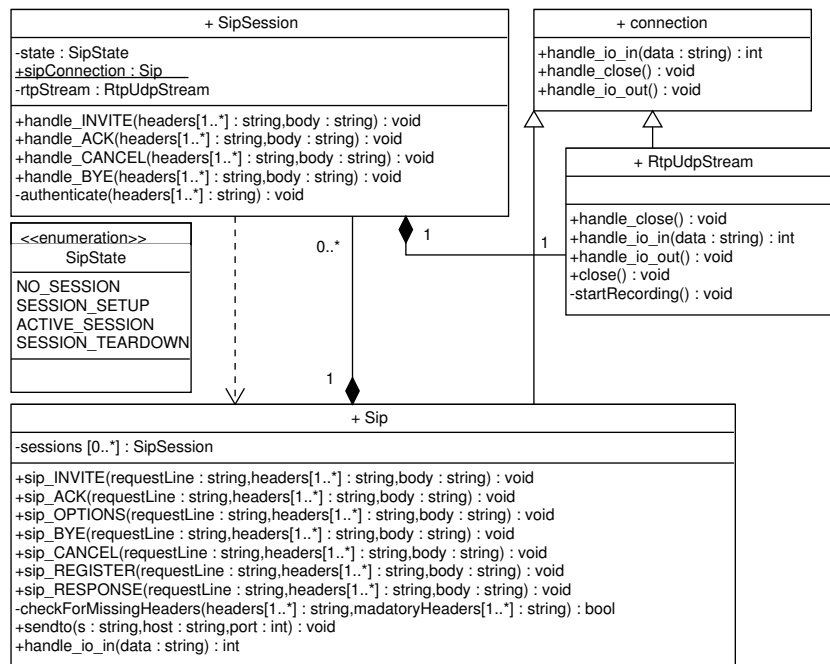


Figure 5.3.: UML class diagram of SIP honeypot module: `connection` is part of `dionaea` and written in C, it exposes an interface to Python using Cython

5.7. Traffic Analysis

Traffic analysis tools examine all or part of the data passing a network node in order to condense the information contained or to filter the data for certain events or signatures. Intrusion detection systems (IDS) scan passing network data for pre-defined signatures or rules. Every hit is logged to a log file or database. One example for an open-source IDS is Snort [10]. IDS potentially produce many hits per minute or even per second depending on the attack surface and size of the network. Therefore, the information provided by the IDS has to be further condensed (and preferably not reduced) before it can be used by human analysts or for automatic decisions.

An example of a Snort rule detecting INVITE flooding attacks looks like this:

```
alert ip any any -> any 5060
(msg: "INVITE scan"; content: "INVITE"; depth: 6;
threshold: type both, track by_src, count 30, seconds 3;
sid: 500001;)
```

More SIP-specific rules are available on the Snort Community Rules website [10].

This results in an alert written to Snort's log file:

```
[**] [1:500001:0] INVITE scan [**]
[Priority: 0]
08/20-01:44:42.603524 192.168.50.3:53 -> 192.168.50.5:5060
UDP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:45
Len: 17
```

5.8. Event Correlation

The aforementioned necessary processing of the information provided by an IDS is done by event correlation engines. A simple and lightweight open-source event correlator is SEC [78]. The correlation decisions are based on predefined rules that match log events using optional time intervals. This is also useful to prevent so-called false positives due to single and isolated events such as failed authentication or a wrong entry on a blacklist. Also, correlating events in real-time opens up the possibility to adapt defense mechanisms as new threats and variations thereof occur.

SEC uses text-based rule files with `key=value` pairs in each line. Among many others, SEC supports the following most important rule types. The matching pattern used for these rules is usually given as a regular expression.

Single Triggered when a log entry matches the pattern. Similar functionality to an IDS examining network traffic.

SingleWithThreshold Triggered when a log entry matches the pattern *and* the pattern has occurred a certain number of times (threshold) within a given time window.

A log entry is also called an event. It does not have to originate from a text log file, even though it is the most common form of primary input. Certain rules can trigger further events as described below. These events can then be processed the same way the first primary events from log files did.

The following list describes actions that can be taken when a rule matches an incoming log entry. Actions can be combined sequentially, for example creating a new SEC event and writing to SEC's log file.

event Creates a new event which can be processed by rules positioned later in the chain. This action is useful for normalising incoming log file entries.

create Creates a new context which will exist for a given amount of time. Contexts are useful in order to avoid processing events in certain situations. For example, a context can be set for the night to ignore matching rules during that time of the day.

add Adds an entry to an internal table. This table can be used to store information for a limited time, and print it to the screen when another event matches.

write Write an arbitrary string to a given log file. The string can reuse elements of the matched pattern.

shellcmd This type of action is needed to pass results from SEC on to other programs. For example, it can be used to send a command to another computer on the network via SSH.

5.9. Software Versions

The exact version or built numbers of the software used to run the experiments of this thesis are given in table 5.1.

Software	Version
VMware Workstation	7.0.0, build-203739 (on Windows XP)
OMNeT++	4.1 (on Debian GNU/Linux)
INET	for OMNET 4.0/4.1, build 20100723
GNUplot	4.4
Snort	2.8.0.2, build 75
SEC	2.5.3
Scapy	2.0
Asterisk	1.4.24
KPhone	4.2
Linphone	2.1.1
Twinkle	1.2

Table 5.1.: Versions of software used for this thesis

5.10. Unsuitable Tools

Apart from the software mentioned above, other tools have been considered or might appear suitable to be included in this thesis. The reasons for the decisions against them are given now. A reason that applies to all categories of software is that of scope: this Master's thesis concentrates on a specific scenario and therefore can not utilise or compare all available possibilities.

5.10.1. Security Information and Event Management (SIEM)

Several options for open-source SIEM software exist such as OSSIM and Cyberoam iView [79, 80]. However, after an examination of their feature list and plugins, both exemplary tools only aggregate incoming data without providing solutions for existing or quickly developing malicious activities on the network. In other words, their functionality is confined to generating reports and raising alarms if thresholds are exceeded. Also, no VoIP-specific features or rules are supplied. Nevertheless, VoIP-specific information management and event management processes could be implemented in one of the available open-source SIEM systems.

5.10.2. Hardware Phones

Because it is impossible to test all available varieties of VoIP SIP phones, the attempt has been made to abstract the end-user's phone as much as possible. This can be achieved by using modular and extendible SIP clients with a programming interfaces for common scripting languages such as Python.

Open-source softphones are much easier to modify and extend than the firmware of hardware phones. Also, they can be monitored more closely from the computer they are running on. A hardware phone usually only exposes the network traffic that is going in and out of it.

5.10.3. Commercial Security Software and Appliances

Commercial software, particularly in the information and network security domain, is often more advanced and receives quicker updates than free software. A common business model for security software developers is to distribute the core component under a free license (such as the GNU Public License) and provide support and updates for professional, that is paying, customers.

This thesis tries to evaluate the performance and suitability of open-source software in large networks. Since the attack scenario is very specific and built in a controlled lab environment, it does not rely on the latest security updates to discover viruses or malicious network traffic. Also, a research project like this can be more flexible and independent when it is not tied to a specific vendor or product. Covering a wide range of commercial products can cost a lot of money when acquiring products or committing to commercial updates, which is not justifiable for a research project of that scope.

6. A Novel VoIP Security Architecture

This chapter describes the main work of this thesis, the novel security architecture for a large SIP-based VoIP network. First, a brief outline of the motivation for such an architecture is given, which is based on the work of the previous chapters. Then, the concept of the security architecture is explained in detail. Scenarios to describe the function of the architecture in certain situations are given at the end of the chapter. The techniques and methods used to implement the specific parts of the architecture concept are described in the next chapter.

6.1. Motivation

In large VoIP networks with many inexperienced users that have not received special training in information security and cannot spend much time on improving the security of their systems, it is often only a matter of time until someone gets infected by a virus or attacked by a social engineering attack. In particular, home users switching to VoIP for their landline connection are likely to simply plug in the box they receive from their provider. Therefore, the company who runs such a large network should put measures in place to protect users from infected hosts on the same network and from getting infected themselves.

6.2. Concept

The goal of this VoIP security architecture is to make each user more secure even if attacks and virus infections are happening. The driving concept behind the architecture is collaboration, that is servers and clients communicate with each other to increase detection rates of malicious activities. On the end-user side of communication this collaboration can be active or passive. An active collaboration involves software that the end-user has to run close to his VoIP equipment in order to collect additional information about malicious activities that the service provider cannot see from his position. In a passive collaboration the user does not have to change his VoIP and network setup or configuration. A small-scale setup including the key components of the architecture as well as examples of user reports are shown in figure 6.1.

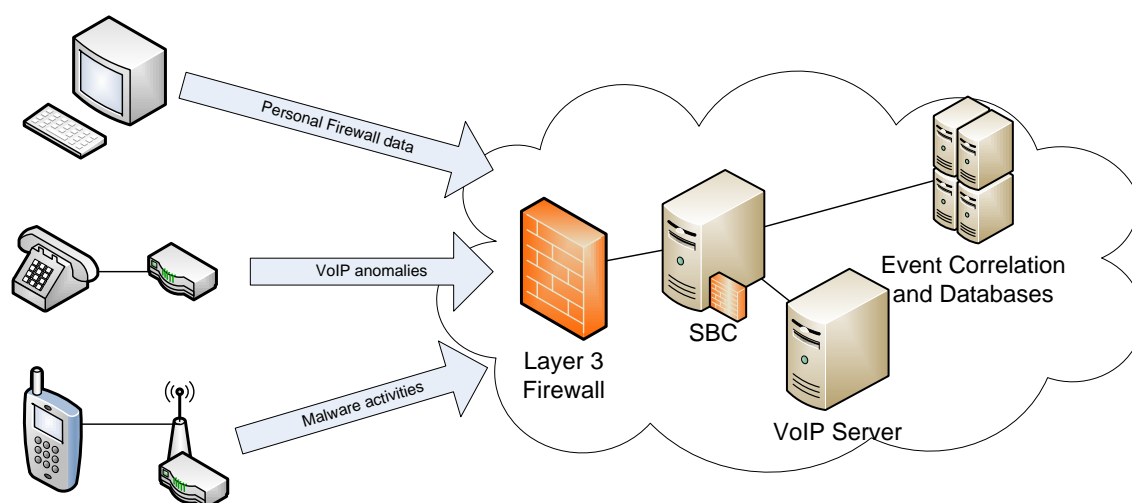


Figure 6.1.: Security architecture inputs: the end-users are shown on the left with their reports sent back to the service provider on the right

Compared to other general purpose architectures that focus mostly on computer systems and attack vectors like websites or email attachments, our VoIP security architecture uses specific characteristics of VoIP traffic such as SIP and RTP packets. Also, the IDS and event correlation rules have to be specifically tailored to VoIP data and networks to defend against the specific threats.

As mentioned at the end of chapter 2, anomaly-free VoIP traffic is relatively easy to predict. Because of that we can use statistical methods to detect anomalies in network traffic and anomalous changes in user behaviour. It has been shown that DoS attacks can be detected by comparing the current network traffic with an estimated baseline [81].

From a single user's perspective, it is much more likely that someone else on the same network gets attacked before the worm or automated attack hits that particular user. Even large-scale attacks launched from many proxies against a large number of users at the same time (a typical botnet attack) are unlikely to reach all users of a network simultaneously. This could be due to different targets (different hardware phone models and softphones) and differences in end-user defense mechanisms. If one assumes that all users run the same risk of getting attacked, then the risk for a single user is

$$P = \frac{n_{attacks} * P_{attack}}{n_{users}} \quad (6.1)$$

where n_{users} can be adapted to reflect only the number of users in a subgroup, for instance the subset of users using a particular VoIP telephone. P_{attack} denotes the probability of an attack on the network and $n_{attacks}$ is the number of attacks within a defined time period.

The concept of a distributed security architecture which provides protection for end-users by collecting and correlating information across the entire network is similar to cloud-based (distributed) antivirus solutions. Because more information is available, the effectiveness of distributed systems is greater. However, a lot of data has to be transferred between hosts and servers which can decrease performance. On devices that do not have the processing power of high-end computer systems, in particular mobile devices, the detection functionality of the network service yields better rates compared to local (offline) analysis [58].

6.2.1. End-Users

Trust and Security

In chapter 3, rootkits are mentioned as a way to place malicious code in a computing device such as a personal computer or a VoIP telephone. This means that some of the intelligence of the system has to be pulled back towards the core of the network (see chapter 2). While it is true that the firmware or operating system on a residential gateway (RGW) can be manipulated in the same way, it offers a more controlled device than the manifold of different phone models or even personal computers with their different operating systems and software versions.

Extending the User's Devices

A piece of security software can be deployed at the end-user to improve security-related communication with the service provider. For example, an intrusion detection system could be installed at the user's residential gateway to collect accurate information about malicious packets directed at the user or leaving the user's broadband connection.

While all of the inputs from the end-user side shown in figure 6.1 can be generated automatically, cases exist where users should be able to report suspicious activities and VoIP anomalies to the service provider. However, it is crucial that a few users are not able to alter the VoIP networks behaviour by introducing malicious or wrong data. This can be achieved by statistical methods such as behaviour change detection and anomaly detection. The implementation of such methods is beyond the scope of this work.

A reporting option for VoIP users can be as simple as pressing a certain number pad key or combination of keys to indicate an incoming malicious call they want to be automatically blocked on the provider side in the future. As soon as enough reports from different users about one particular kind of call are transmitted to the provider, an anomaly detec-

tion/event correlation engine can be used to decide whether the VoIP network is in need of protection from this attack.

6.2.2. NAT

In order to provide useful log data, the collaborating RGW has to send information back to the correlation server that makes it possible to distinguish every individual user. From a service provider's point of view, this can be done using the user's public (facing the Internet) IP address because all (non-private) IP addresses have to be unique. If two nodes share the same IP address, routing behaviour is undefined. Therefore, the IDS on the user's RGW has to be configured to only listen on the external interface because all the NAT operations (translating the IP address of the phone on the internal network to the public address) have been done when a packet leaves the RGW on the external interface. Also, for packets sent to the user, *no* NAT has occurred when they enter the RGW on the external interface.

6.2.3. Detecting VoIP Phone Crashes

Several methods can be applied to detect phones that change from an "up" state (operational) to "down" (unreachable, not operational). SIP phones have to register with a PBX before the user is able to send and receive phone calls. This registration has to be renewed in regular intervals, otherwise the PBX assumes that the phone went offline. By choosing a short registration interval, a phone that recently crashed can be quickly detected. However, this usually requires the user of a phone to set a very low registration interval.

Because the security architecture should be deployable without too much coupling to the end-user's existing hardware and software, the previously described solution to detect phone crashes is not suitable. Instead, a separate process running on the user's gateway

to his ISP can provide the same information without any changes in the phone's configuration.

The crash of a particular UA (hardware phone behind a RGW or softphone) can be detected by utilising a heartbeat program that is deployed between the Internet uplink and the phone. In case of a hardware phone, the heartbeat monitor could be located on the RGW; in case of a softphone, the monitor might have to run on the same computer if it is connected directly to the service provider. Nevertheless, a RGW with the required security software is recommended because it is the least intrusive on the user's computer equipment.

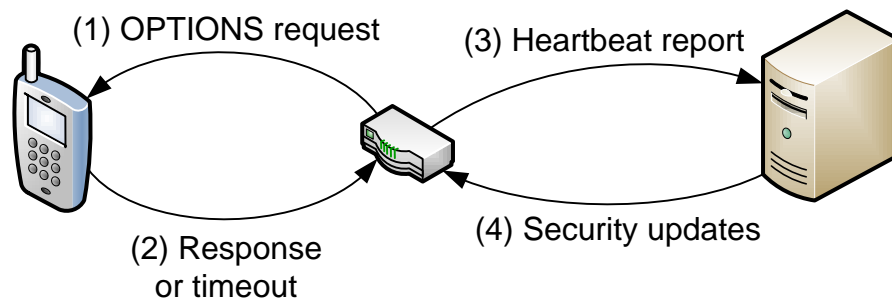


Figure 6.2.: Heartbeat monitor: (1) sending a request to the phone to check its status; (2) receiving a reply (phone alive) or experiencing a timeout (phone not responding); (3) sending report about inactive phone to event correlation server; (4) receiving security updates in the form of blacklists or user notifications

Different methods for such a heartbeat “ping” are possible but using SIP OPTION requests via UDP packets is considered the quickest and less intrusive way for this setup. The reason for this is that almost all SIP-based VoIP devices support this request which can be sent at any time and does not interrupt ongoing VoIP sessions. The Asterisk PBX for example sends out an OPTIONS request to each registered phone once a minute (by default using the Asterisk NOW distribution). As soon as the device does not answer the request anymore for a given amount of time, the heartbeat monitor assumes that it crashed or has been switched off.

Disadvantages

Because the heartbeat monitor cannot distinguish between phone crashes and deliberate shutdowns, several problems remain that have to be solved. Those special scenarios and possible solutions are listed below.

1. If a phone is unplugged (in case of a hardware phone) or turned off (in case of a softphone or the computer running the softphone), the heartbeat monitor will report the new state to the event correlator. This could lead to a flood of “crash” reports at certain times such as in the evening. A blacklisting approach for such times can eliminate this problem, however protection provided by the security architecture is decreased during those excluded times of the day. Also, by combining deregistration information from the PBX with reports from the heartbeat monitor, normal shutdowns can be detected and filtered.
2. If a phone enters the down state because of the user’s action, the heartbeat monitor should be notified. This way, it will not report the missing heartbeat response as a crash. Yet this method is more intrusive since it requires an extension of the hardware phone’s firmware or the softphones software on the computer, respectively.
3. If blacklisting times and feedback from the phone is not possible, too many phone shutdowns might trigger an alarm in the event correlator. The time windows and thresholds of the specific event correlation rules can be adapted to the VoIP network in question, thus avoiding false positives. However, this depends highly on the specific characteristics of the network and might take a long time to set up properly.

Supported Phones

The following open-source softphones have been tested with the OPTIONS request “ping”: Twinkle, KPhone, and Linphone (see table 6.1).

Phone	Response	Allow SIP Header
KPhone	200 OK	INVITE, OPTIONS, ACK, BYE, MSG, CANCEL, MESSAGE, SUBSCRIBE, NOTIFY, INFO, REFER
Twinkle	404 Not Found	not available
Liphone	200 OK	not given in response

Table 6.1.: Results of OPTIONS request against different phones

Note that the 404 Not Found SIP response message is an answer from a VoIP device that indicates a running SIP server. Therefore, the heartbeat monitor can be sure that the phone is still running and responding.

6.2.4. User Notification

One effective way to inform users of attacks and vulnerabilities is to use the same communication channel as the threat itself. The reason for that is that no additional form of communication has to be established which can save costs and makes deployment of VoIP much easier if a lot of UAs are involved. On the other hand, a DoS or flooding attack will not only render the VoIP service useless but it will also block the user notification channel.

Several methods and ways to inform the user of problems and malicious activity over an existing VoIP architecture are possible:

1. Voice channel, sending information as spoken, pre-recorded announcements
2. Text channel, sending written information on the hardware phones' display or to the user interface of a softphone

Option 1 offers seamless integration into every possible VoIP architecture since only the central VoIP server has to be modified. On the other hand, option 2 might be more user-friendly and provides a constant flow of information without calling the user.

SIP can be extended with instant messaging capabilities using the MESSAGE method [82]. It makes sense to use SIP for user notifications because it can be guaranteed that

all VoIP devices can receive the notifications. However, most phones will not be able to process these messages and only open-source softphones can be extended to support instant messaging. Many softphones already include presence and messaging features, for instance KPhone for Linux (see figure 6.3).

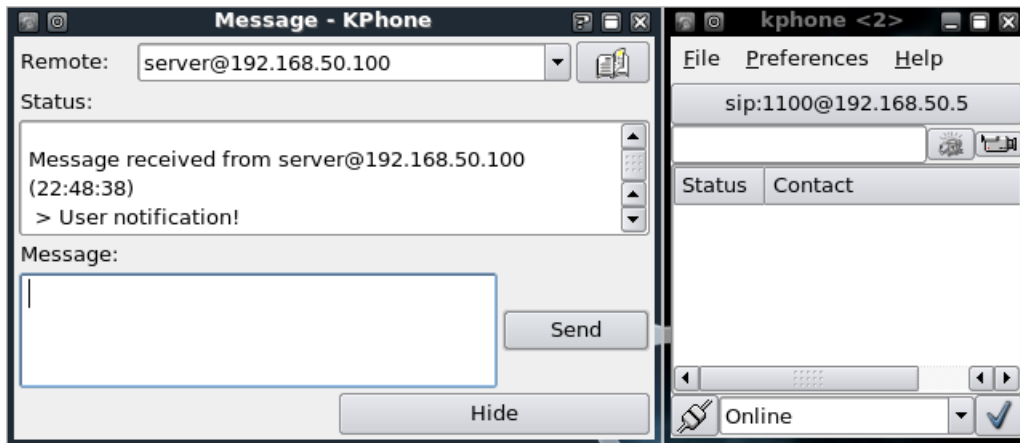


Figure 6.3.: User notification in KPhone

The “Rogue Antivirus” Phenomenon

Faked antivirus programs (fake or rogue AV) with names like “SystemSecurity”, “SystemGuard”, or “Xpantivirus” are a specific type of rougueware and an increasingly dangerous problem of network security [83]. Malicious hackers write software that looks similar or even exactly like popular legitimate antivirus programs such as Microsoft Security Essentials. Its intention is to make the victim believe that he is infected with one or several different viruses. The user is prompted to install the fake AV and pay for the license in order to clean the computer. However, this actually installs malware on the computer. This kind of malicious software is called scareware. In consequence, users cannot trust notifications anymore that recommend to install AV software to protect the computer.

User notifications that are sent from the service provider to the end-user over the Internet face the same problem of missing credibility. However, in contrast to in-browser pop-ups, authenticated SIP MESSAGE requests are more trustworthy. Nevertheless, as long

as packets are not encrypted or signed, man-in-the-middle attacks can always alter the content of SIP packets. This is due to the fact that authenticated SIP messages merely carry a challenge response hash in the authentication header line. The rest of the message is not protected in a cryptographic and secure way.

6.3. Privacy Concerns

Of course any attack countermeasure that holds back information from a user raises privacy concerns. Also, refusing services such as a connection to the VoIP network could result in a customer demanding a compensation. Therefore, it is important to get the user's approval before an analysis and filtering of any traffic takes place. A motivation for users is the increased security as shown in chapter 8 and notification of current and new threats as described earlier in this chapter.

6.4. Scenarios

This section describes exemplary situations where the novel security architecture provides additional security through the mechanisms specified in the first part of this chapter.

6.4.1. Scenario 1: SIP Scans

SIP scans are used to discover devices on a computer network before an actual attack is launched. An additional feature is fingerprinting these devices, that is using information from the scan such as the content of a response packet to determine the version of the operating system or application that is running on the device.

Most SIP requests can be used to scan a network for VoIP devices. The most common one is an OPTIONS request because it usually does not cause any reaction on the targeted phone. The only evidence of such a scan is a log entry which is almost never looked at by

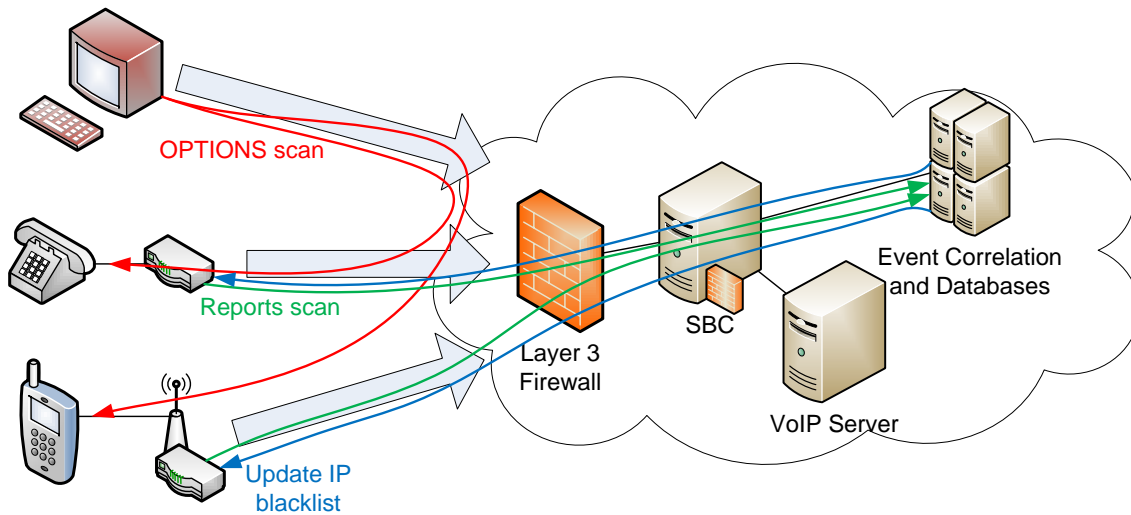


Figure 6.4.: Scenario 1: Reaction to a SIP OPTIONS scan

the user. An OPTIONS request also sends back more information about the capabilities of the phone compared to other request types.

The reaction of the components of the security architecture are (see figure 6.4):

1. First end-user's RGW reports an incoming scan;
2. second end-user's RGW reports an incoming scan with the same parameters;
3. the event-correlator recognises the organised scan against the network even though the single end-points only detect one scan instance each;
4. the service provider sends out updates to the end-users so that they can block further malicious scans or attacks (blacklisting).

6.4.2. Scenario 2: Phone Crash

Most applications connected to a computer network are vulnerable to some kind of remote exploit attack. One example is KPhone for Linux in versions 4.1 and earlier¹. The exploit works because these versions of KPhone do not verify message length headers against

¹<http://www.securiteam.com/unixfocus/5PP0B1FCLY.html>

the actual length of the content (a STUN message in this case). This leads to a memory pointer pointing to a location outside the received message thus crashing the application. Examples of attacks against hardware phone are given in [37].

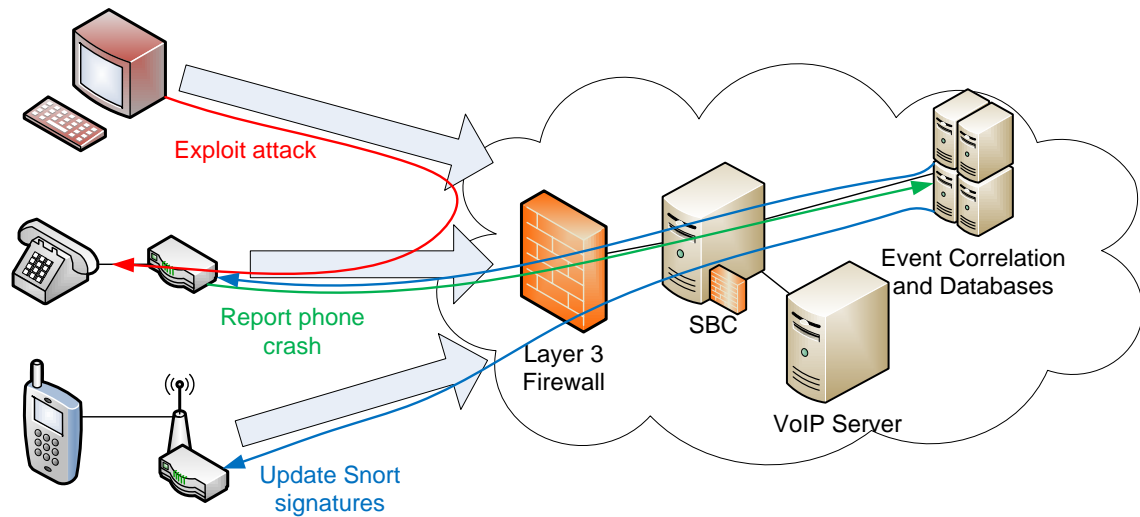


Figure 6.5.: Scenario 2: Reaction to an exploit attack that crashes the phone

If certain conditions hold in the target, sophisticated attacks are able to take control of the application or the operating system, either as a normal user or as administrator/root (through a process called privilege escalation). However, most of the time the application will merely crash. If this happens very often, for instance because it is an ongoing attack that sends the same malicious packets to the target with high frequency, it can result in a successful DoS which renders the phone virtually useless.

A heartbeat monitor in the end-user's private network continuously "pings" the phone which can be either a hardware phone or a softphone running on a computer. This way, a crashed phone can be detected. A report is generated by the RGW and sent to the event correlation engine as shown in figure 6.5. Possible reactions to such an attack include sending an update for the IDS from the event correlator to all users.

6.4.3. Scenario 3: User Reports

A VoIP server such as Asterisk can respond to incoming signals. Most of the time, these are dual-tone multi-frequency signals (DTMF) which are used to dial a number on analogue phones. During a SIP session these DTMF signals can be intercepted, written to a log file, and analysed by the event correlation engine.

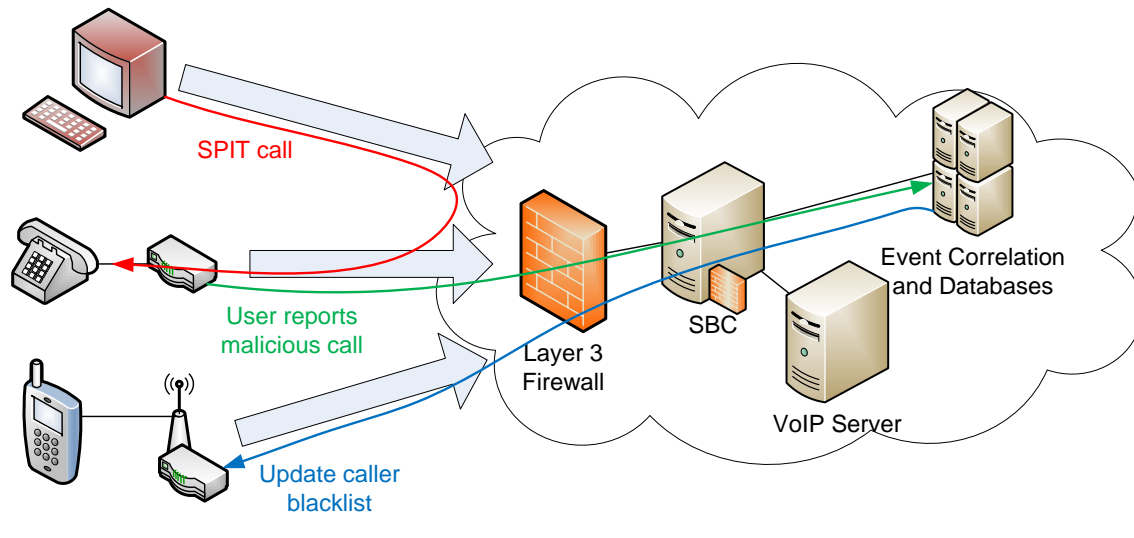


Figure 6.6.: Scenario 3: Reaction to a SPIT call

This scenario assumes that a set of end-users receive a SPIT call and report it to the service provider using the phone's interface. The reporting of the malicious call and the reaction by the event correlation engine on the service provider's side of the network are shown in figure 6.6.

7. Implementation

This chapter describes the implementation details of the experiments conducted for this thesis. First, the implementation and setup of the testbed is given. This is used to test the feasibility of the tools described in the previous chapter. Then, an explanation of the network simulations using OMNeT++ follows. As [84] points out, “performance and scalability” are the major advantages of using a network simulation framework for information security purposes.

7.1. Proof-of-Concept Testbed

The testbed is used to demonstrate the concept of the novel security architecture described in the previous chapter on a small scale using a realistic network environment. This is achieved by using one Linux operating system for each end-user (softphone plus security software). Instead of individual computers for each operating system, virtual machines are created using VMware Workstation (see chapter 5). After a quick outline of the tasks and steps involved in setting up the testbed, more detailed explanations of the implementation are given.

7.1.1. Testbed Security

In order to prevent server and client software, such as VoIP registrars and packet generators, running on the virtual machines from affecting computers and other devices out-

side the testbed, a secure and controlled network boundary has to be established. The testbed's access to the university's network and the Internet is controlled by an Internet gateway running iptables firewall software on a Debian GNU/Linux system. Nodes in the private network of the testbed can only access the Internet if the gateway is running. The gateway also has to grant access before individual machines obtain a route to the Internet. This is done by sending a ping ICMP packet from the gateway to the machine that wants to access the Internet. Internet access is needed mostly for system maintenance reasons, for example to install new software which does not come pre-installed with the distribution.

7.1.2. Collect IDS Data at Residential Gateway

End-users usually have a router deployed at the perimeter of their network. These devices can execute many different tasks, such as establishing a connection to the service provider, providing a VoIP adapter to analogue telephones, and wireless access point functionality. A device that combines those mechanisms is called a residential gateway (RGW). Today's RGWs run on modern and fast hardware that enables them to provide many different features in parallel. Often, Linux or Unix-based embedded operating systems are installed. The event collection system is implemented using Linux software which can be deployed on routers that are capable of running Linux or customised firmware.

The key events that are collected by customer-side RGWs in VoIP networks and that are correlated at the service provider side are:

1. Simultaneous SIP scans against a certain number of users,
2. known malformed packets (Snort rules), and
3. crashes of UAs or RGWs.

The first two events are realised using the open-source IDS Snort. The rules that are used to scan the incoming network data can be adapted to new attacks.

The third event can only be detected with the help of an additional active tool. A Heartbeat monitor written in C (see appendix D for the relevant source code) scans the VoIP devices in its network in regular intervals. As soon as a device is not reachable anymore and the OPTIONS request times out, the heartbeat monitor on the RGW tells the correlation server of the service provider about the event.

7.1.3. Securely Send Collected Data to Service Provider

An easy way to establish an authorised and encrypted communication channel between end-users and the centralised correlation and decision engine is the use of the Secure Shell (SSH) protocol. This secure channel can then be used by the event correlator to retrieve heartbeat reports and to send out security updates or user notifications.

In a large-scale deployment of the security architecture, transmitting information through SSH might have to be changed to a different technology because of performance requirements (TCP and SSH handshakes involve many packets to initiate a session) and concurrency issues (writing information to a file using an SSH remote command). If a more stable secure communication channel with strong authentication mechanisms such as a public key infrastructure (PKI) is desirable, a virtual private network (VPN) such as IPsec or OpenVPN should be used.

7.1.4. User Notification on Different Phones

The SIP MESSAGE request was chosen as the method used to deliver notifications to the hardware or software phones of the end-users [82]. Unfortunately, since it is not a feature that is commonly used and since most people do not associate landline telephones with text messages or instant messaging, support for this SIP request is often incomplete. Examples of several phones are given in table 7.1. The MESSAGE requests for these tests were generated with Scapy. The code can be found in appendix B.

Asterisk is also capable of sending notifications to phones using the *SendText* command [85]. This is done by adding a configuration like

```
exten => 123, 1, Answer
exten => 123, 2, SendText(hello world)
exten => 123, 3, HangUp
```

which will send a text notification to the phone when the extension number 123 is called.

Phone	Support	Source
Twinkle	does not respond	Testbed
KPhone	supported	Testbed
Linphone	supported	Testbed
Snom hardware phone	limited support	Voip-Info.org [85]
Cisco hardware phone	supported	Voip-Info.org [85]

Table 7.1.: Support for SIP MESSAGE request

7.1.5. Event Correlation Rules

An event correlation engine can take several of the events mentioned in the last section and put them in a temporal context. If a certain number of events during a predefined time interval occur, an alarm or a reacting action is triggered as shown in figure 7.1.

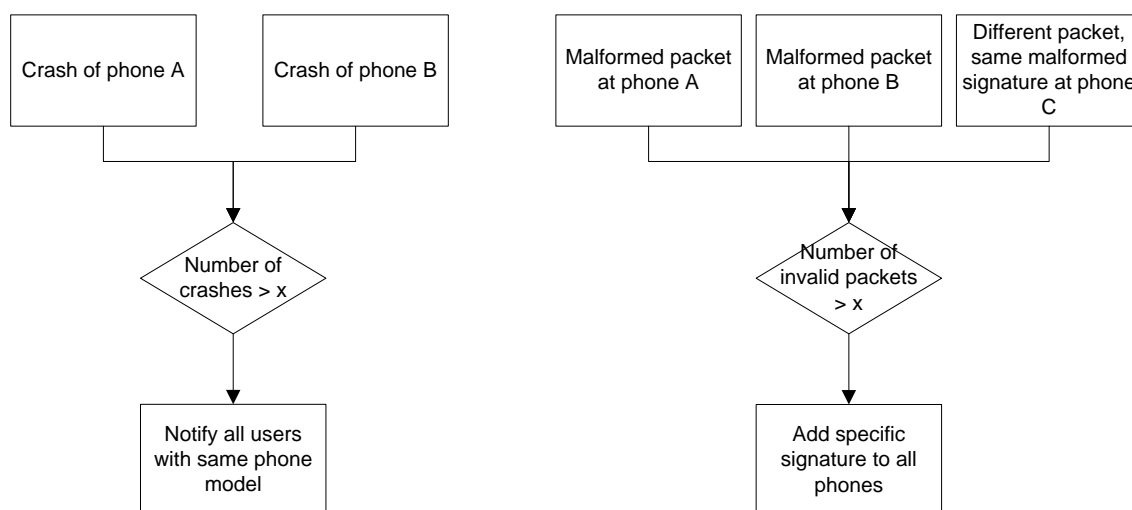


Figure 7.1.: Two event correlation examples

The following list gives examples of actions taken after certain events affecting several end-users have been detected. The variable x denotes the number of end-users on the network. This can be changed to adjust the algorithms to the specific network.

1. More than x end-users report a malicious traffic event from the same origin: add origin to blacklist and notify all users.
2. More than x end-user report a crash of their phone's software: add signature to IDS of all users.
3. End-user reports connection attempt from blacklisted IP: "upvote" IP on central blacklist.
4. End-user reports blacklisted malicious data signature: "upvote" signature on central blacklist.
5. More than x end-users manually report a SIP session (a conversation) as spam or malicious: add origin or extracted signature to blacklist and notify all users.

See figures 7.2 and 7.3 for diagrams of the event correlation rules. All the rules used in SEC can be found in text in appendix A.

Snort has to be restarted if its signatures change. This can be done by sending a SIGHUP signal to the snort process on the RGW.

7.1.6. Phone Crashes and Phones Unregistering

The event correlation engine can distinguish between an UA unregistering from the PBX (Asterisk) and an UA crashing due to a software bug or exploit attack. Usually, the heartbeat monitor running on the RGW will report any missing response to the SIP OPTIONS request (see table 6.1). Due to the simple implementation of the heartbeat monitor in the testbed (see appendix C for the relevant code sections), it cannot determine the reason

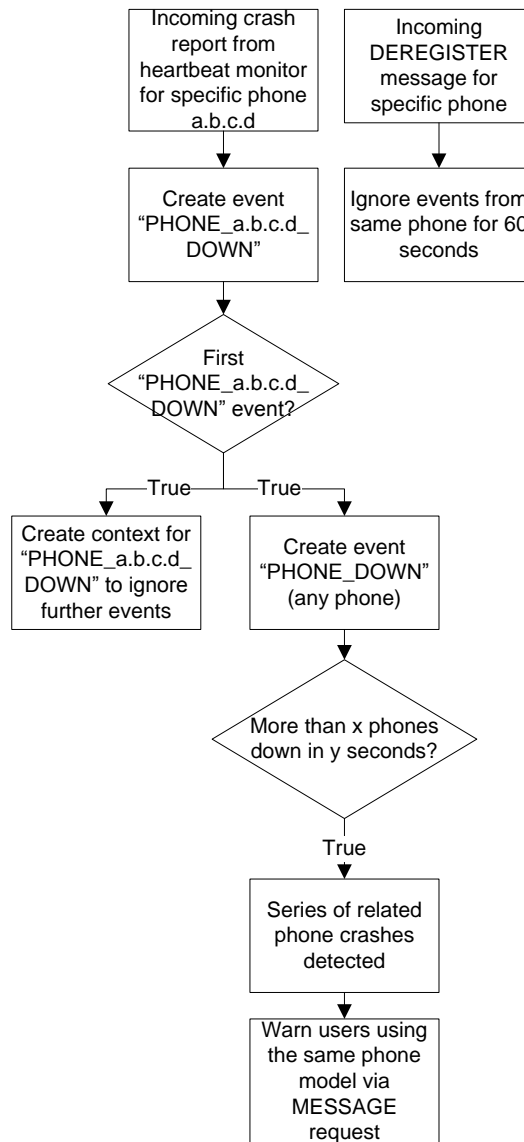


Figure 7.2.: Event correlation in SEC: crash detection

why the phone is not responding. However, the combination of Snort and heartbeat monitor on the RGW, and the event correlator SEC makes it possible to ignore phones not responding if they unregistered from the PBX.

An excerpt of the Snort log of one RGW as sent to the correlation server is given below. SEC can use this information to ignore incoming heartbeat timeout reports because the event correlation engine knows that the UA exited normally and did not crash suddenly. This is done with a simple correlation rule that ignores incoming heartbeat reports from

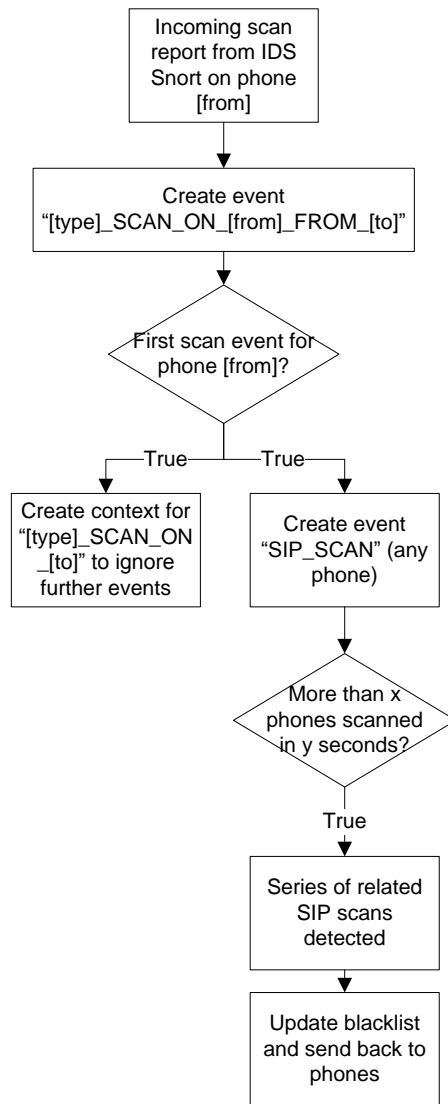


Figure 7.3.: Event correlation in SEC: scan detection

the RGW as long as the phone is not registered at the PBX. The phone is identified by its public IP address.

```

Nov 1 00:20:36 192.168.50.11
snort: [1:2:0] DEREGISTER {UDP}
192.168.50.11:5060 -> 192.168.50.5:5060

```

7.2. Network Simulation

In order to obtain data about the performance of the security architecture on a larger scale than is possible within the testbed, network simulations using OMNeT++ (introduced in chapter 5) are conducted. A simulation framework also allows for quick development of new features without having to patch a real-world application or even operating system or firmware. Additional features can only be provided if the source code is available (open source) or if the software uses a plugin or module architecture. Additionally, it is easy to collect data from sensors across the network and inside individual modules, and aggregate them in a central data file.

Figure 7.4 shows a screenshot of a small VoIP network and a detailed view of a home user's network with RGW and UA.

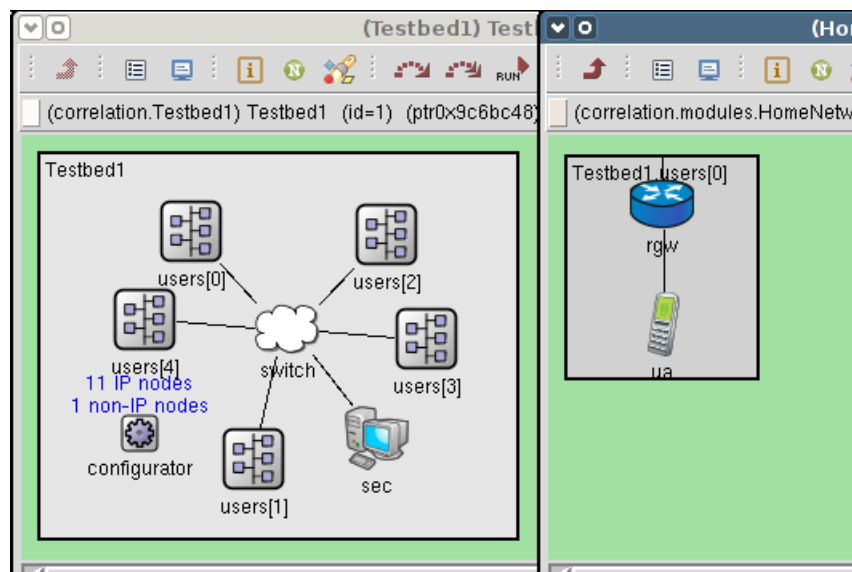


Figure 7.4.: Small simulation (overview on the left, detailed view of a home user's network on the right)

7.2.1. Limitations and Assumptions

Simulations can only produce satisfying results if the underlying model is accurate and as realistic as possible for the scenario to be analysed. However, it is not feasible to implement details such as different media (copper cable, fibre optic, wireless) or even their underlying physical properties when modeling Internet connections. Therefore, realistic assumptions are made. These assumptions can be based on previous experiments (empirical) or on calculations (theoretical).

The assumptions for the network simulation model of the security architecture are as follows:

Internet Connection Speed and Bandwidth

INET for OMNeT++ provides very accurate implementations of many network protocols such as Ethernet, IP, TCP, and UDP. However, processing times on devices and propagation delays for transmissions across the network are not simulated automatically. For the purpose of evaluating the performance of the security architecture, common round-trip times (RTT) that are encountered by Internet users are used. For private (no high-performance business or university) Internet lines, RTTs are usually between 10ms and 40ms for national connections as shown in table 7.2.

Target	minimum	maximum	average
www.canterbury.ac.nz	11.25ms	30.72ms	14.42ms
www.auckland.ac.nz	23.95ms	42.15ms	30.38ms
www.google.co.nz	23.74ms	37.26ms	27.82ms
www.kit.edu (Germany)	312.86ms	337.27ms	318.92ms
github.com (USA)	245.72ms	257.87ms	250.11ms

Table 7.2.: Round-trip times for different servers from private ADSL in New Zealand (10 pings with a packet size of 64 bytes per target)

Processing Speed of RGWs and Event Correlator

As mentioned above, processing times of devices or protocol stacks are not automatically implemented in OMNeT++ and INET. However, processing delays can be realised using timers that send internal messages to notify the OMNeT++ module that a process has finished.

VoIP and Non-VoIP Attacks and Vulnerabilities

In order to implement realistic attack scenarios, the simulation model has to contain many software parts and processes, for instance all the execution paths inside the vulnerable software. This is not feasible for an architecture that does not focus on one specific application or exploit. Therefore, a special attack packet is used that simply indicates the kind of attack. The RGWs and UAs react accordingly when they receive such a packet.

7.2.2. Implementation Steps in OMNeT++

The following paragraphs describe the steps necessary to develop a functional network simulation model using OMNeT++ and INET.

Module Descriptions

Before the VoIP network can be assembled for the simulation model, its parts have to be defined and implemented. Some of these parts come with OMNeT++ and INET, for instance an ethernet switch and a network host running TCP and UDP applications. Most modules have to be developed specifically for the simulation of the security architecture. In OMNeT++, modules that do not contain other modules are called simple modules. Modules that contain other modules are called compound modules. All modules that are defined specifically for the simulation of the security architecture are listed below.

1. UA UDP application: simple module embedded in INET's standard host
2. SEC UDP application: simple module embedded in INET's standard host
3. Application firewall module with routing functionality to inspect traffic on all network layers, derived from INET's standard host
4. RGW UDP application: simple module embedded in application firewall compound module

While modules 1 and 2 only have to implement the behaviour of the according UDP application, module 4 is embedded in a new module (module 3). This application firewall and router module is not supplied by INET. It is needed for forwarding ARP requests and IP packets like an INET router module while still being able to run UDP applications such as the RGW UDP application that handles incoming SIP traffic.

Network Description

After all parts of the network are defined (either by OMNeT++, INET, or in new modules), the VoIP simulation network is assembled. The network features an arbitrary number of end-users, each one consisting of an RGW and a UA. All network traffic (incoming and outgoing) for an end-user's UA has to pass through the RGW. The two devices are connected via a 100Mbps Ethernet LAN, a setup found in most private and business local networks.

All end-users are interconnected via a central router. The connection between the users and the central router is a line with a random delay within the boundaries of real-world Internet traffic delay mentioned above. To avoid ARP requests for what is a very basic routing setup, the Point-to-point protocol (PPP) is used. This resembles production DSL connections which are usually connected via PPP or a variation thereof such as PPPoE (chapter 3.6.2 [36]). The event correlation engine (simple event correlator, SEC) is also connected locally to that switch. Therefore, the switch can be seen as the simplified network infrastructure of the service provider running the VoIP service.

Modules and networks can be defined either by using a graphical user interface or by writing the specification in the special-purpose scripting language NED. All module and network specifications are compiled into the executable simulation binary.

Custom Message Specification

OMNeT++ provides a simple mechanism to develop new message formats without forcing the developer to write all its functionality in C++. The definition of a SIP packet in the simulation does not include all the mandatory headers specified in the RFC [82] but concentrates on information that is essential to simulate VoIP attacks and their mitigation techniques. The definition of the SIP message type is given in table 7.3. Table 7.4 shows the fields of the attack message type used to send abstract attack descriptions from the attacking UA to the victim UAs. Tables 7.5 and 7.6 show the structure of the supporting attack report and security update message types.

Field name	Type	Description
msg	String	Request method or response code
from	String	SIP “From” header
to	String	SIP “To” header
via	String	One SIP “Via” header
callId	String	SIP “Call-ID” header
seqNo	Integer	Sequence number part of SIP “CSeq” header
seqMethod	String	Request method part of SIP “CSeq” header
contentLength	Integer	SIP “Content-Length” header
body	String	Embedded SIP body, for example an SDP message

Table 7.3.: SIP message definition for the network simulation

Field name	Type	Description
type	Integer	0 = no attack, 1 = OPTIONS scan, 2 = exploit attack
payload	String	Attack-specific payload, for example exploited vulnerability

Table 7.4.: Attack message definition for the network simulation

Field name	Type	Description
type	Integer	0 = IDS, 1 = heartbeat
userId	Integer	End-user identifier
payload	String	Report payload, for example phone's program version

Table 7.5.: Report message definition for the network simulation

Field name	Type	Description
type	Integer	0 = IP blacklist, 1 = signature update
payload	String	Update payload, for example IP address or signature

Table 7.6.: Update message definition for the network simulation

Behaviour Implementation

When all networks, modules, and message types are specified, the behaviour of the new modules has to be implemented. In the OMNeT++ framework, this is done using the programming language C++. A C++ file has to be provided for each newly developed simple module. Compound modules are not programmed in C++, they merely consist of input and output gates, parameters, and simple modules. A UML class diagram of these C++ classes is shown in figure 7.5.

The classes in the first two layers from the top are supplied by OMNeT++ and INET, respectively. `VoipModule` is used to generalise common functionality of UAs and RGWs UDP application classes. One example for such a function is displaying a notification message next to the affected end-user's network in the simulation GUI.

`IpWithBlacklist` replaces INET's default IP module in order to add blacklisting functionality. The extended IP module is used in the RGWs which receive updates for their blacklists from the SEC.

Now, the network simulation can be visually executed at different speeds using the OMNeT++ GUI. Notification areas (bubbles) above affected networks or devices indicate important events as shown in figure 7.6. Additional messages and all events of a simulation run are also written to log files. If performance is more important than visual

feedback, simulations can also be run using the command line which requires less processing power.

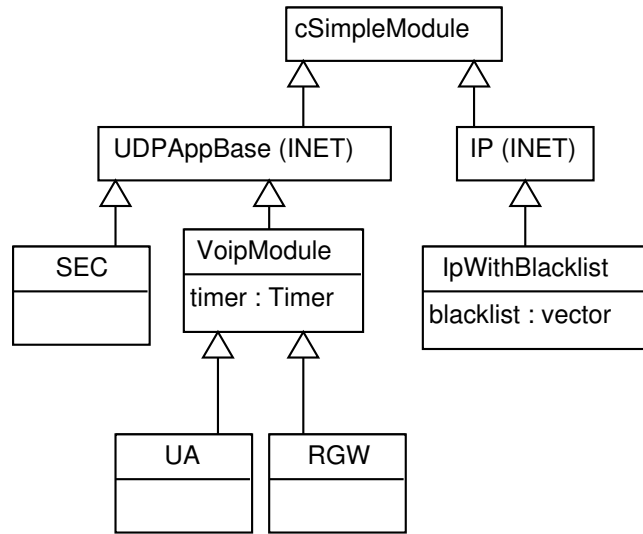


Figure 7.5.: UML class diagram of simulation modules

Scenario Configuration

In order to maintain flexibility while running different simulated scenarios, OMNeT++ utilises an INI file that contains key/value pairs. Each module (simple or compound) has a set of defined parameters that can be set either in the NED file of the module (see Module Descriptions) or in the simulation configuration file (INI file). When a simulation is started, one or a set of scenarios with different parameter settings can be selected. The values of the parameters can be retrieved inside the C++ source code of a module by using the OMNeT++ API.

In order to create a functioning simulation with INET, the configuration file has to be used to assign UDP application modules to the host modules. In the simulations conducted for this thesis, there was only one UDP application per host running on port 5060 (SIP).

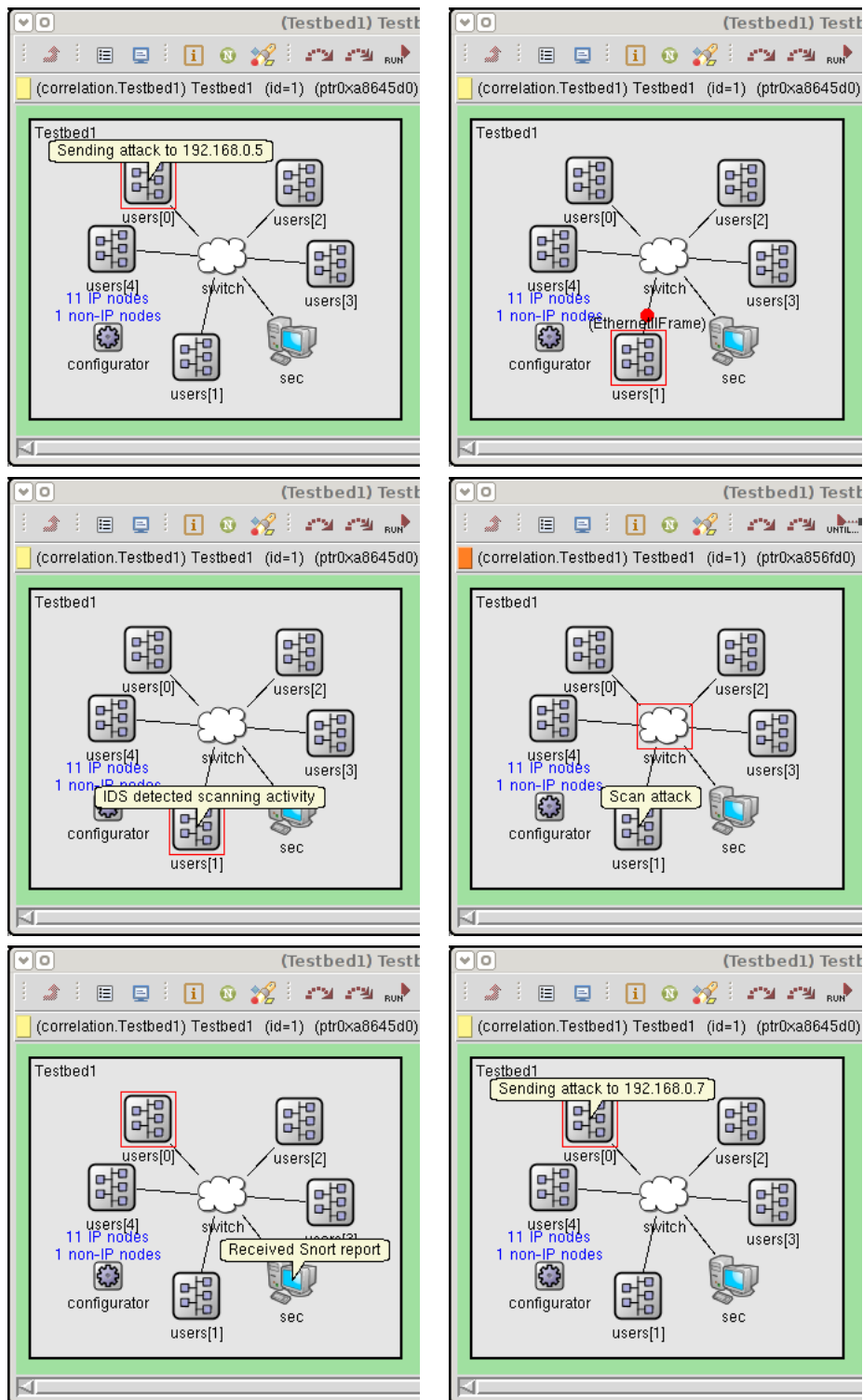


Figure 7.6.: Sequence of screenshots of a simulated OPTIONS scan attack

8. Evaluation

The previous chapters described the novel security architecture. In this chapter, the results from testing the system for security enhancements and performance are presented. First, an overview of the evaluation process is given. Then, VoIP honeypot results, qualitative results from the proof-of-concept testbed, and quantitative results from the network simulation are analysed. Known and yet to solve problems encountered during the experiments are described at the end of this chapter.

8.1. Evaluation Process and Input Data

In order to determine the usefulness and effectiveness of the implementation of the new architecture, measurements of the security improvements have to be taken. The qualitative and quantitative methods used in the evaluation part of this thesis are described in the following sections.

Two different types of input data can be used for the security and performance experiments: constructed data and network traffic from the real world. The former can be traffic specifically generated to exploit a known vulnerability, or it can be randomised using fuzzing techniques. VoIP-specific network traffic from the real world has been captured and analysed using a honeypot as described in chapter 5. Based on the results from the honeypot and from other reports on malicious VoIP traffic, specific attacks (OPTIONS request scanning) were generated for the testbed. The network simulation utilised abstract

attack messages that do not resemble real attack data. The behaviour of these messages were based on observations of the honeypot and the testbed.

8.2. Honeypot Results

The VoIP module for the honeypot Dionaea collected data over the period of 48 days. During that time, 70 SIP connections to port 5060 of the honeypot server were made from the outside. Figure 8.1 shows the distribution of connections per day for the period of observation (from 1/11/2010 to 18/12/2010). The number of connections per day ranges from 0 to 5. The average number of connections per day, or mean, is $\mu = \frac{70}{48} \approx 1.46$, the median is 1. This can also be seen in figure 8.2 which shows that the most common case is one connection per day. Also, there is a clear clustering around the mean whereas high values are very rare. The standard deviation for the number of connections per day is

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2} \approx 1.93 \quad (8.1)$$

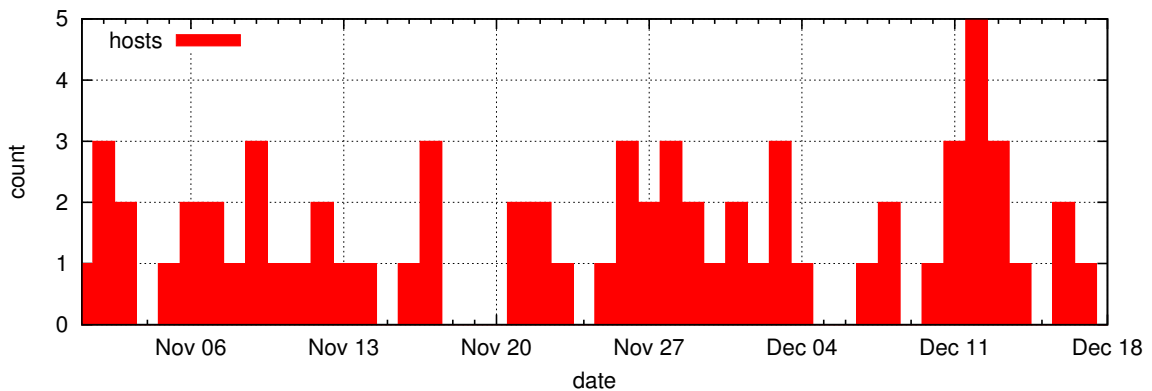


Figure 8.1.: Number of hosts connecting to VoIP honeypot on port 5060 between 01/11/2010 and 18/12/2010

The distribution of originating countries of the remote hosts is shown in table 8.1. It does not come as a surprise that connections from China make up for more than half the scans. USA follows with a large distance and the other countries only contributed one or two

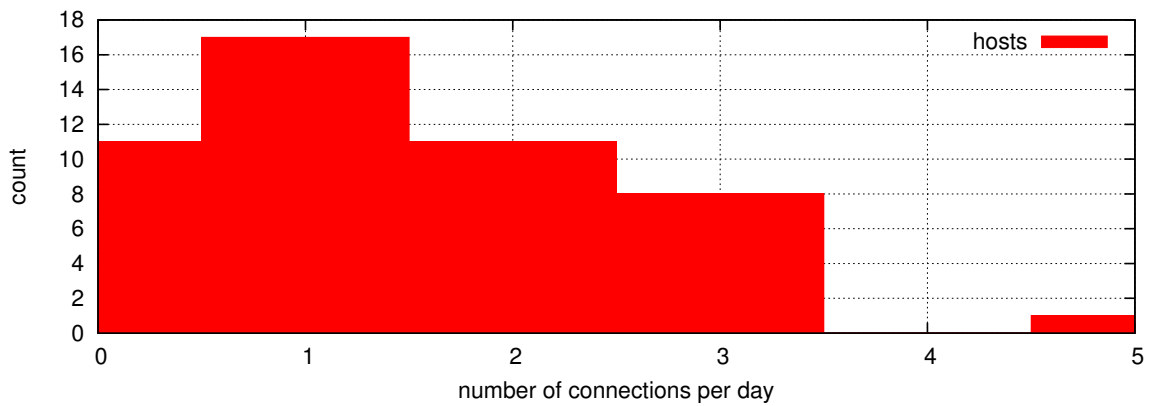


Figure 8.2.: Frequency of connections per day to VoIP honeypot

connections each. With exception of the continent of South America, the connections originated from all over the world.

Country	Connections	Percentage
China	41	58.6%
USA	10	14.3%
Japan	2	2.8%
Korea	2	2.8%
Nigeria	2	2.8%
Russia	2	2.8%
Australia	1	1.4%
Brazil	1	1.4%
Germany	1	1.4%
Spain	1	1.4%
Hong Kong	1	1.4%
Italy	1	1.4%
Netherlands	1	1.4%
Romania	1	1.4%
Thailand	1	1.4%
Taiwan	1	1.4%
South Africa	1	1.4%

Table 8.1.: Honeypot connections by country (percentages rounded)

An example of one malicious incoming SIP request from the scanning tool sundayddr and one response from the honeypot is given at the end of this chapter.

8.2.1. Similarities

There are strong similarities between most of the connections made by scanning tools to the SIP honeypot. All of these used a long random integer for the Call-ID. However, all connections made by the sundayddr tool contained a Call-ID starting with 003484. The Call-IDs of the other tool resembled a more random pattern. Also, all incoming OPTIONS requests used the extension number 100 for the From and To parameters of the SIP header.

The different characteristics of scanning hosts connecting to the VoIP honeypot are shown in table 8.2. The groups are based on common characteristics found in several scan connections. Many connections share the same values across all header fields.

User agent	Via	From
Asterisk PBX	127.0.0.1	“sipvicious” <sip:100@1.1.1.1>
sipvicious	real address or 127.0.0.1	“sipvicious” <sip:100@1.1.1.1>
sundayddr	192.168.1.9	“sipsscuser” <sip:100@192.168.1.9>

Table 8.2.: Notable characteristics of connecting SIP clients sending OPTIONS requests

Only one SIP request from SIPVicious used a different Contact header:

```
sip:None@127.0.1.1:5060.
```

8.2.2. SIP Calls

Two incoming SIP connections tried to establish a SIP call by sending an INVITE request. These two connection attempts occurred on November 30 and December 8. That is about 2.8% of all total connections to the SIP service. The order of the SIP header lines were the same as with SIPVicious and sipssc. However, different extensions were used for the To and From headers. Both INVITE requests were sent to the extension number 0014089078000. This number does not appear to be random because seven out of 13 digits are zeros. It looks like a telephone number with a country code 0014 or +14 which is the prefix for numbers in Brazil. The extension number used for the sender of

the request is 9309565. According to GeoIP [86], those two call attempts originated from Amsterdam in the Netherlands and San Francisco, USA.

However, a response for the INVITE requests was not sent because of a software bug in the honeypot module. The moment the incoming INVITE request is processed by the SIP parsing function, the following log entry is generated:

```
warning: Mandatory header content-type not in message
```

This due to the fact that a phone call has to specify the details of the session, which is usually done by embedding an SDP message in the SIP body. Because of the missing SDP message, no research on brute-force attacks against the authentication mechanism used for INVITE requests could be conducted. Also, no RTP channel was established that could have provided useful information on the type of the phone call.

8.2.3. Conclusion

In conclusion, most connections came from either the original or a modified version of SIPVicious. As shown in chapter 4, the signatures of the SIP messages are very similar to those of the VoIP tool suite SIPVicious.

8.3. Security Improvements of the Security Architecture

Many existing programs and security mechanisms have proven that they are suitable for improving network security in VoIP environments. In particular, open-source software is very flexible when it comes to supporting new protocols or implementing new communication paths between several different programs. One example of this is the collaboration of the IDS Snort and the event correlator SEC. The combination of the simulated IDS reacting to incoming special attack packets, and the simple event correlator (SEC) as a

dedicated server within the network simulation provides enhanced security for the virtual VoIP network. The scenarios that have been used to test the capabilities of the novel security architecture are as follows.

8.3.1. Detection and Mitigation of SIP Scanning Attacks

Snort is able to detect all known attacks as soon as signatures are available. Therefore, for a single user Snort in combination with an adaptive firewall can provide a significant improvement in security. This is valid as long as the user does not fall victim to a zero-day exploit. Statistically it is more likely that one of the many other users gets attacked first. As soon as the victim of the attack triggers the distribution of new Snort signature files, all the other users on the network will be safe from this particular attack.

SEC has proven to be a solution that is easy to deploy in small networks. However, large enterprise networks often demand additional features such as load-balancing. SEC does not provide these features out-of-the-box but it is feasible to adapt it to the requirements of the specific environment.

An improvement in end-user security can be achieved by notifying them of current threats. Often, users forget about persistent threats or are not aware of new techniques used by criminals and malicious hackers.

Figure 8.3 shows a section of the simulation event log during one successful OPTIONS scan. Most of the layers of the TCP/IP networking stack involved are hidden to simplify the display of the message flow through the network. The attacking user (user 3) is only shown from an outside perspective, that is as seen by the service provider. The receiver of the OPTIONS scan is shown in more detail, that is UA and RGW are shown as separate axes. Because UA and RGW communicate via Ethernet, an ARP request and response dialog can be seen in the centre of the screenshot.

Figure 8.4 shows a section of the simulation event log from the last OPTIONS scan report to the update of the blacklist of malicious IP addresses of all the users' RGWs. The

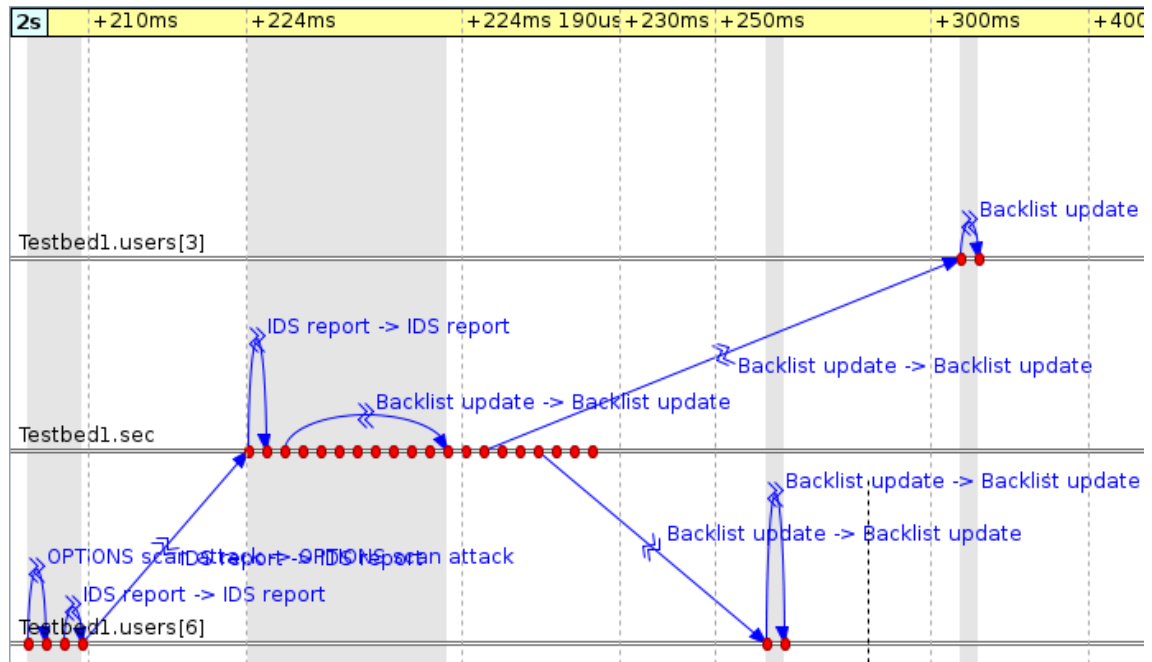


Figure 8.4.: Section of simulation event log showing blacklist update from SEC to all users. User 6 sent last OPTIONS scan report that exceeded the threshold of the event correlator. X axis shows time in nonlinear format.

reported scan attacks, the SEC updates the blacklist of all users. The attacker ignores the blacklisted IP address of its phone so that he can continue sending malicious OPTIONS scans. All other RGWs drop packets from the blacklisted IP address. Therefore, the flow of scan reports stop after six reports.

Figure 8.6 shows the same scenario with four attackers. Each attacker starts sending OPTIONS requests at a random time in the interval $[0.5s, 2s]$ with uniform probability distribution. The delay between each scan attack is a random time interval out of $[0.3s, 0.8s]$ with uniform probability distribution. The graph shows the sum of OPTIONS requests sent by the attackers, the sum of detected scans by the end-users' RGWs, and the number of scan reports at the SEC over time.

Figure 8.7 shows a larger and more realistic VoIP network scenario with 100 users. Now, the SEC resets the counter for the scan reports after the threshold has been reached. This is needed to be able to block more than only one IP address from a malicious user who is sending OPTIONS requests to the rest of the network. This leads to a sawtooth plot,

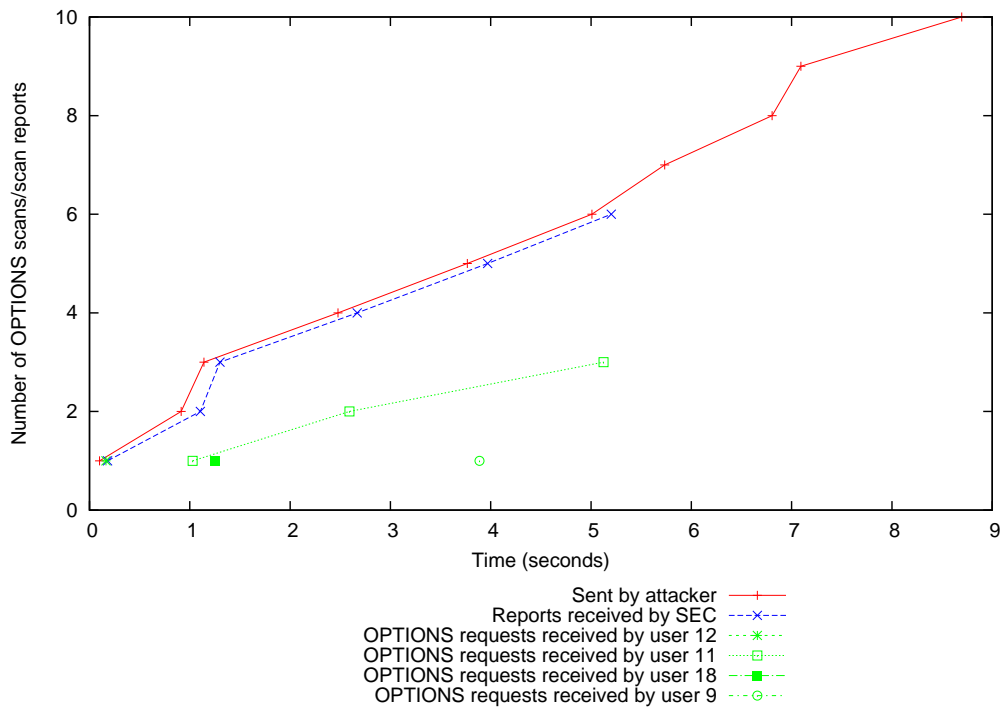


Figure 8.5.: Scan attacks over time, one attacker, showing the number of packets sent by the attacker, total scans received by the affected users, and scan reports at SEC

and to more and more attackers being blacklisted on the RGWs of the regular users. The derivative of the scan reports curve decreases over time. This means that the curve showing the number of scan reports grows slower over time. This shows that the event correlator and the IP blacklists on the RGWs provide an effective countermeasure against SIP scan attacks.

8.3.2. Detection and Mitigation of Exploit Attacks

After malicious OPTIONS requests, exploit attacks against vulnerabilities in the software of specific phones were examined. The scenario of exploit attacks against vulnerabilities in the VoIP phones of multiple users yields similar results and patterns compared to the

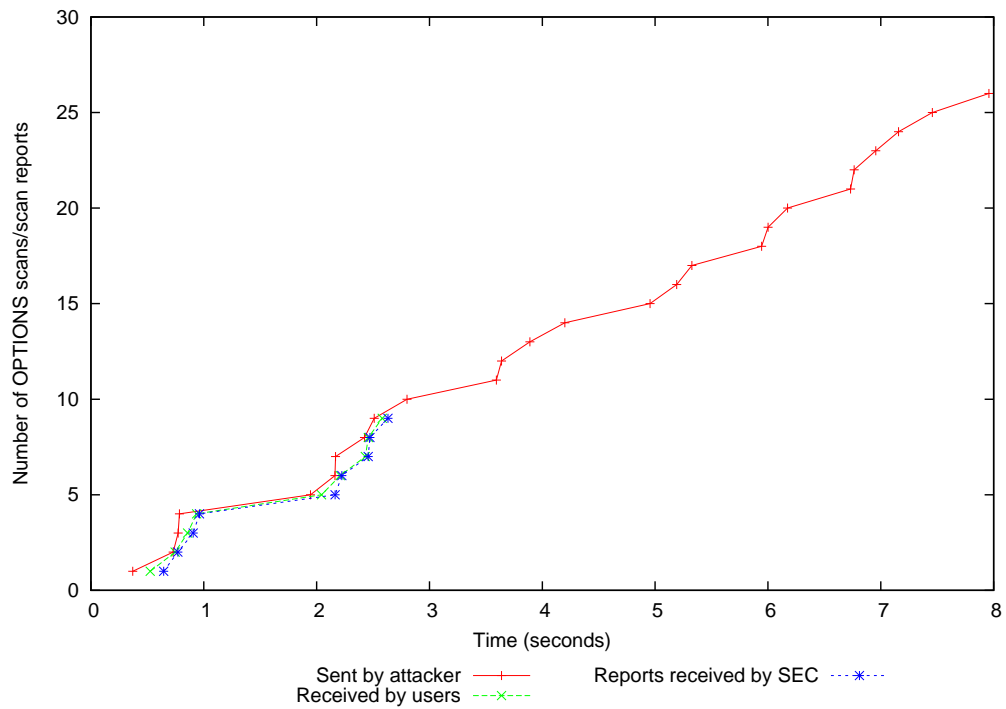


Figure 8.6.: Scan attacks over time, multiple attackers, showing the number of packets sent by attackers, total scans received by the affected users, and scan reports at SEC

previous configuration with SIP scans. However, the mechanisms that mitigate the attacks work in a different way. The heartbeat monitor that checks the health of the phone from the RGW proved to successfully detect sudden phone software crashes. The testbed showed that it can also distinguish between phones unregistering from the PBX and actual crashes.

Figure 8.8 shows the packets sent by the four attacking users, the sum of successful exploit attacks at the victims' phones, and the sum of signature matches after the SEC received crash reports from multiple RGWs.

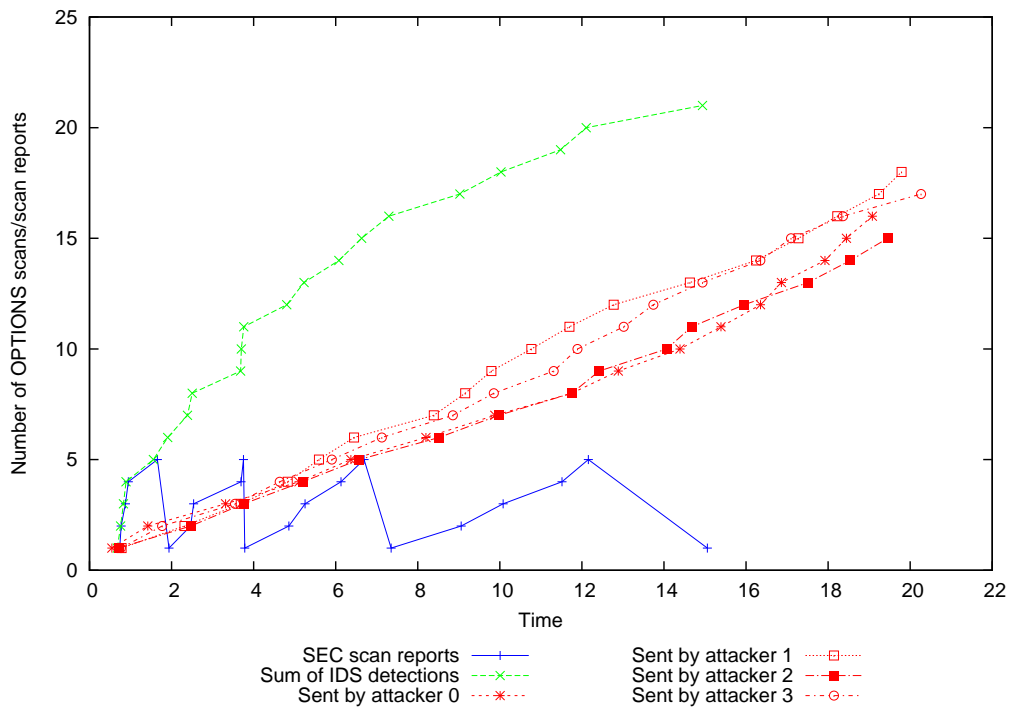


Figure 8.7.: Scan attacks over time, multiple attackers, large network

8.3.3. Discussion of RGW and SEC Parameters

Several simulation runs with the same scenario and pseudo-random numbers were used to test the influence of different settings of core parameters of the security architecture. These parameters are the delay between SIP OPTIONS requests for the heartbeat monitor and the threshold for phone crash reports in the RGWs and SEC, respectively. The values used for these parameters along with the time of the first successful exploit signature match of the IDS in the RGW after signature updates have been sent out by the SEC are given in table 8.3 and figure 8.9. The conclusion from these experiments is that the heartbeat interval is far more important to the reaction time of the architecture than the threshold. This is due to the fact that at a fast rate of attacks the threshold is exceeded

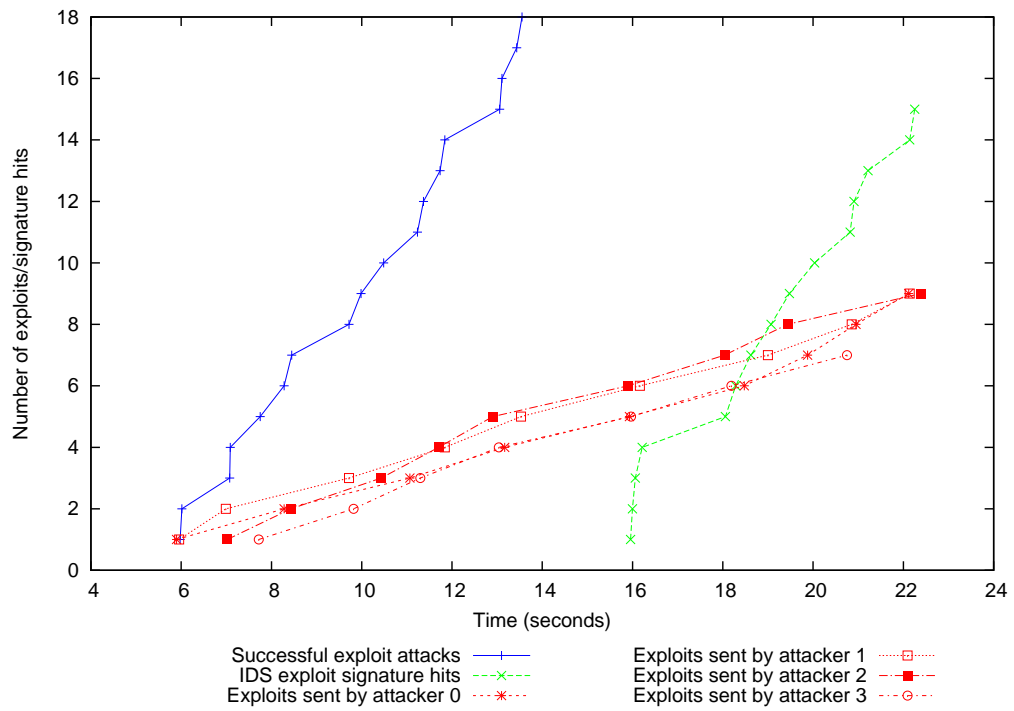


Figure 8.8.: Exploit attacks and signature hits over time, multiple attackers

very quickly but reports are only sent out after a ping of the heartbeat monitor times out, that is no response comes back from the phone.

Heartbeat interval (RGW)	Reports threshold (SEC)	Time of first signature match (RGW)
10	5	21.69s
10	10	21.69s
10	50	37.22s
30	5	37.22s
30	10	37.22s
30	50	37.22s
60	5	71.47s
60	10	71.47s
60	50	71.47s

Table 8.3.: Parameters used for heartbeat interval and phone crash reports threshold and time of first successfully matched exploit signature

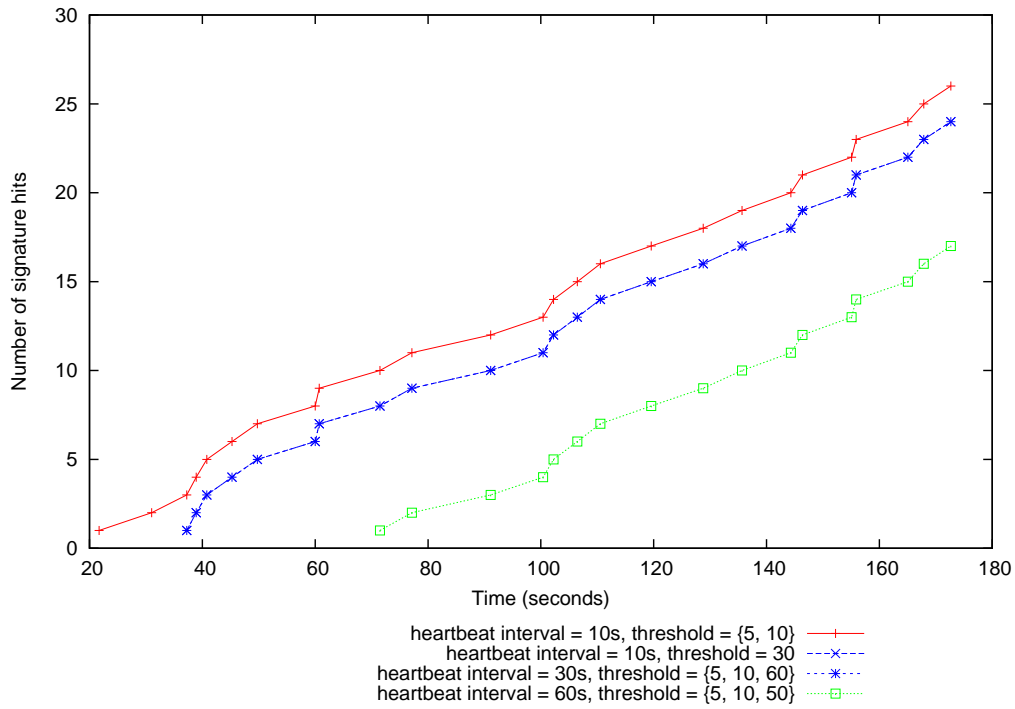


Figure 8.9.: Comparison of simulation runs with different parameter settings for heartbeat interval and cash reports threshold. Note that the reaction time of the security architecture is mostly dependent on the heartbeat interval, not the reports threshold used in the SEC.

8.4. Performance

A simulation framework like OMNeT++ can be used to obtain indicative and conclusive performance data of an implementation of the security architecture. This work describes the results of a proof-of-concept implementation. These results indicate that the security architecture provides its services to the end-users with good performance and little overhead. Further improvements in performance can be made using multicast functionality of IPv6 when sending the same content to many different users. Since the VoIP network is already capable of delivering a high number of packets with real-time requirements, the additional packets for reports, updates, and user notifications are not likely to con-

gest the network. However, in order to determine the limits of the architecture in case of a widespread DoS attack, a thorough analysis and more experiments of such a scenario have to be conducted. This should also include more realistic simulation modules that emulate software from the real-world.

8.5. Known Problems

Limited Range of Supported Phones

Only a limited number of phone models have been tested within the security architecture. This is acceptable since this thesis describes and implements the concept of a novel security architecture for VoIP networks. Also, the network simulation used abstraction layers to be independent from specific attacks or software vulnerabilities by using an attack packet that triggers a certain reaction inside the UAs and RGWs.

Software Only Tested in Lab Environment

Some of the software, preexisting SIP servers and clients as well as newly written software like the heartbeat monitor, provide useful and fully functional features to the section architecture. However, their current implementations can fail or crash in real-life, large-scale networks. One example is the heartbeat monitor that does not respond for a given timeout period if a port is not reachable. In this case, it does not produce an error message that the event correlator could process.

Limited Emulation of Software in Simulation

In order to simplify the simulation of existing software like Snort, SEC, and iptables (Linux firewall), abstraction layers such as attack packets and generic report packets were

used instead of specific malicious SIP packets or realistic Snort log messages, respectively. However, in a real-world scenario, the transmission and processing of this information can have strong implications on reaction time and the security of the network.

Limited Functionality of Honeypot Module

The honeypot module was able to parse and respond to simple OPTIONS request which are commonly used in scanning attacks. More sophisticated requests like an INVITE message can be parsed if they are correct according to the SIP specification. However, honeypot has to be able to react to uncommon and malicious packets. Therefore, a more resilient and realistic honeypot or an improved version of the dionaea honeypot module should be used in further research.

8.6. SIP Request and Response Example

```
stream = [  
    ('in', b'OPTIONS sip:100@132.181.19.2 SIP/2.0\r\n  
Via: SIP/2.0/UDP 192.168.1.9:5060;  
    branch=z9hG4bK-50189216;rport\r\n  
Content-Length: 0\r\n  
From: "sipsscuser"<sip:100@192.168.1.9>;  
    tag=71549043630862915906794138894448007632323469000\r\n  
Accept: application/sdp\r\n  
User-Agent: sundayddr\r\n  
To: "sipssc"<sip:100@192.168.1.9>\r\n  
Contact: sip:100@192.168.1.9:5060\r\n  
CSeq: 1 OPTIONS\r\n  
Call-ID: 129635798199182432847141937\r\n  
Max-Forwards: 70\r\n\r\n'),  
  
    ('out', b'SIP/2.0 200 OK\r\n  
Via: SIP/2.0/UDP localhost:5060\r\n  
To: "sipsscuser"<sip:100@192.168.1.9>;  
    tag=71549043630862915906794138894448007632323469000\r\n  
From: 100 <sip:100@localhost>\r\n  
Call-ID: 129635798199182432847141937\r\n  
CSeq: 1 OPTIONS\r\n  
Contact: 100 <sip:100@132.181.19.2>\r\n  
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE\r\n  
Accept: application/sdp\r\nAccept-Language: en\r\n\r\n')  
]
```

9. Conclusion

9.1. Summary

This thesis described the situation of threats and vulnerabilities in SIP-based VoIP environments. Existing solutions and ongoing research in the area of VoIP network security and threat mitigation were described and analysed for their effectiveness in a scenario where compromised or malicious users are expected.

The results from the proof-of-concept testbed indicate that a security architecture for large VoIP networks can be implemented using only Unix-based systems and free open-source tools. The goal of network security happening in the background without overwhelming either users or administrators with log entries and security-related warning messages was achieved. Beyond the implementation done for this work, areas of potential additions to the architecture, and therefore further improvement of security, were identified. These areas are described in the following section about future work.

The results from the network simulation are indicative but not conclusive for large and complex VoIP networks like they exist in the real world. This is due to the fact that many components of the simulation model were approximated instead of fully emulated. Experiments were conducted to acquire realistic parameters for these components, for example round trip times of data packets on the Internet and the way specific VoIP phones respond to given input. However, not all components could be fitted with realistic settings.

This work showed the effectiveness of virtualisation technology and network simulations for implementing and evaluating small-scale networks and large-scale network models. Simulations were used to automate scenarios and acquire sensor data from many different nodes of the modelled network. Since experiments have to be repeatable and yield useful – that is not random – results, automation was an important factor in designing the testbed and even more in implementing the network simulations.

The VoIP module for the honeypot software Dionaea was developed in cooperation with members from the HoneyNet Project. During the data collection phase, it worked flawlessly for scan attacks using OPTIONS requests. A significant amount of data was gathered, in particular when considering that it ran only on one small subnet that was not advertised. The collected data shows heavy use of mostly related variations of SIP scanning tools that try to discover vulnerable versions of phone models and VoIP servers. However, no SIP or RTP-based exploits or attacks were discovered. Two promising attempts to establish a call by sending an INVITE request could not be put to further use because of missing responses sent back from the honeypot. Therefore, a dialog with the attacker could not be established, which led to loss of potentially important data that goes beyond common SIP scans.

9.2. Future Work

9.2.1. Performance

Because the testbed of the architecture was a proof-of-concept implementation using tools that enabled the thesis' author to demonstrate the use of existing open-source software, performance was not always considered the highest priority. If custom-built software is used instead of many separate tools, and if interpreted programs and scripts are replaced by faster running binary executables, a performance gain on a large network is likely. This

needs further research beyond the proof-of-concept and early indications provided in this thesis.

9.2.2. Realistic VoIP Data

To capture real-world malicious VoIP network data, a honeypot for SIP was written and deployed on the university network and exposed to the Internet. Because the honeypot was confined to one public subnet, its results are not extensive enough to be representative of the whole Internet. However, comparisons and data exchange with other parties who ran the same honeypot setup confirmed the attacks on the honeypot used for this thesis. The area of VoIP honeypots, their deployment, and how to advertise them on the Internet is an important part of future research on a VoIP-specific security architecture.

9.2.3. SIP and RTP Attacks Beyond Primitive Scans

The honeypot used in this work could detect and log scanning attacks consisting of OPTIONS requests. However, actual SIP calls (INVITE requests) and RTP data could not be analysed. This was due to the fact that the incoming INVITE requests did not contain an SDP message embedded in the SIP body. Improvements on the honeypot and the environment in which it is deployed have to be done to get more useful data on dangerous threats. Targeted advertisement of the honeypot's address on the Internet could also increase the rate of incoming messages and therefore improve the results.

9.2.4. Extensions and Generalisation

This thesis' scope was confined to research, analyse, and improve security of a large VoIP network. The main protocols of this thesis, SIP and RTP, can be used in many other applications such as video conferencing, instant messaging, and gaming. With further research

and more experiments, the security architecture can be extended to those applications and even to a general security architecture for all network traffic.

The testbed and network simulation can also be extended to provide a more accurate model of the real world. In particular, the network simulation in this thesis generated messages to trigger certain functionality in the emulated security tools. In a more sophisticated model, an IDS module within OMNeT++ could analyse incoming network traffic and emulate Snort's behaviour.

9.2.5. Incorporating Asterisk

While the open-source Linux distribution “Asterisk NOW” was used to provide a VoIP PBX to the softphones used in the testbed, Asterisk itself was not specifically configured or modified to support the security architecture. Asterisk runs with a very flexible configuration that allows VoIP users to initiate actions specified within Asterisk. One example is the implementation of a user reporting function using DTMF, that is dialing numbers during an active phone call session. Furthermore, extensions on the Asterisk server could provide additional security without (additional) modifications on the end-user's side of the connection.

Bibliography

- [1] Lothar Fritsch et al. “A Holistic Approach to Open Source VoIP Security: Results from the EUX2010SEC Project”. In: *International Journal of Advances in Security* 2.2 (2009), pp. 129–141.
- [2] Georgia Tech Information Security Center. “Emerging Cyber Threats Report for 2009”. 2008.
- [3] Shawn McGann and Douglas C. Sicker. “An Analysis of Security Threats and Tools in SIP-Based VoIP Systems”. In: *Proceedings of the 2nd Workshop on Securing Voice over IP*. Washington DC, USA, 2005.
- [4] Dimitris Geneiatakis et al. “SIP Security Mechanisms: A state-of-the-art review”. In: *Proceedings of the 5th International Network Conference (INC 2005)*. Samos, Greece, 2005, pp. 147–155.
- [5] Angelos D. Keromytis. “Voice-over-IP Security: Research and Practice”. In: *IEEE Security & Privacy* 8.2 (2010).
- [6] J. Oquendo. *VoIP Abuse Project*. Accessed 04/10/2010. 2010. URL: <http://www.infiltrated.net/voipabuse/>.
- [7] Elisabeth D. Zwicky, Simon Cooper, and D. Brent Chapman. *Building Internet Firewalls*. 2nd ed. O’Reilly, 2000.
- [8] Eric Maiwald. *Network Security: A Beginner’s Guide*. McGraw-Hill, 2001.
- [9] Marcus Goncalves. *Firewalls: A Complete Guide*. McGraw-Hill, 2000.

- [10] Snort. *Snort :: Home Page*. Accessed 18/05/2010. URL: <http://www.snort.org>.
- [11] Dinei Florencio and Cormac Herley. “A large-scale study of web password habits”. In: *Proceedings of the 16th international conference on World Wide Web - WWW '07* (2007), pp. 657–665.
- [12] E. Rescarlo. *SSL and TLS: Designing and Building Secure Systems*. Addison Wesley, 2000.
- [13] Christopher Soghoian and Sid Stamm. “Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL”. In: *Social Science Research Network (SSRN)* (2010).
- [14] William Stallings. *Data and Computer Communications*. 8th ed. Pearson Education Inc., 2007.
- [15] Asaf Shabtai et al. “Google Android: A State-of-the-Art Review of Security Mechanisms”. In: *Arxiv preprint arXiv:0912.5101* (2009).
- [16] A.D. Schmidt et al. “Monitoring smartphones for anomaly detection”. In: 14.1 (Nov. 2009), pp. 92–106.
- [17] J. H. Saltzer, D. P. Reed, and D. D. Clark. “End-to-end arguments in system design”. In: *ACM Transactions on Computer Systems* 2.4 (Nov. 1984), pp. 277–288.
- [18] Henning Schulzrinne et al. *RFC3550: RTP: A Transport Protocol for Real-Time Applications*. Accessed 15/01/2011. 2003. URL: <http://tools.ietf.org/html/rfc3550>.
- [19] M. Handley and V. Jacobson. *RFC2327: SDP: Session Description Protocol*. Accessed 15/01/2011. 1998. URL: <http://tools.ietf.org/html/rfc2327>.
- [20] M. Handley, C. Perkins, and E. Whelan. *RFC2974: Session Announcement Protocol*. Accessed 15/01/2011. 2000. URL: <http://tools.ietf.org/html/rfc2974>.

- [21] Gonzalo Camarillo. *SIP Demystified*. McGraw-Hill, 2002.
- [22] Eve M Schooler and Stephen L Casner. “A Packet-switched Multimedia Conferencing System Conferencing System”. In: *ACM SIGOIS Bulletin* 1.1 (1989), pp. 12–22.
- [23] Henning Schulzrinne. *Simple Conference Invitation Protocol*. Accessed 13/12/2010. 1996. URL: <http://tools.ietf.org/html/draft-ietf-mmusic-scip-00>.
- [24] Counterpath. *X-Lite*. Accessed 01/12/2010. 2010. URL: <http://www.counterpath.com/x-lite.html>.
- [25] Skype. Accessed 23/08/2010. URL: <http://www.skype.com>.
- [26] Jon Hardwick. *Session Border Controllers: Enabling the VoIP Revolution*. Accessed 15/01/2011. 2005. URL: <http://www.metaswitch.com/download/sessionbordercontroller.pdf>.
- [27] J. Rosenberg et al. *RFC3261: SIP: Session Initiation Protocol*. Accessed 15/01/2011. 2002. URL: <http://tools.ietf.org/html/rfc3261>.
- [28] M. Baugher et al. *RFC3711: The Secure Real-Time Transport Protocol (SRTP)*. Accessed 15/01/2011. 2004. URL: <http://tools.ietf.org/html/rfc3711>.
- [29] K. Egevang and P. Francis. *RFC1631: The IP Network Address Translator*. Accessed 15/01/2011. 1994. URL: <http://tools.ietf.org/html/rfc1631>.
- [30] J. Rosenberg, R. Mahy, and P. Matthews. *RFC5389: Simple Traversal Utilities for NAT*. Accessed 15/01/2011. 2008. URL: <http://tools.ietf.org/html/rfc5389>.
- [31] Mihai Constantinescu, Daniel Gheorghica, and Victor Croitoru. “Secure and Flexible Method for SBC/Firewall Management”. In: *International Symposium on Signals, Circuits and Systems (ISSCS)*. 2009, pp. 1–4.

- [32] Isaac Lee and Ray Hunt. “A novel design of a VoIP firewall proxy to mitigate SIP-based flooding attacks”. In: *International Journal of Internet Protocol Technology* 3.2 (2008), pp. 128–135.
- [33] Konrad Rieck et al. “A Self-Learning System for Detection of Anomalous SIP Messages”. In: *Second International Conference on Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*. 2008, pp. 90–106.
- [34] Jens Fiedler et al. “VoIP Defender : Highly Scalable SIP-based Security Architecture”. In: *Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications*. 2007, pp. 11–17.
- [35] B. Ramsdell and S. Turner. *RFC5751: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2*. Accessed 15/01/2011. 2010. URL: <http://tools.ietf.org/html/rfc5751>.
- [36] Andrew Tanenbaum. *Computer Networks*. 4th ed. Prentice Hall, 2003.
- [37] David Endler and Mark Collier. *Hacking Exposed VoIP*. McGraw-Hill, 2007.
- [38] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. 4th ed. Prentice Hall, 2006.
- [39] What’s My Pass. *The Top 500 Worst Passwords of All Time*. Accessed 15/01/2011. 2008. URL: <http://www.whatsmypass.com/the-top-500-worst-passwords-of-all-time>.
- [40] Tom’s Hardware. *Your Top 20 Most Common Passwords*. Accessed 15/09/2010. 2010. URL: <http://www.tomshardware.com/news/impervarockyou-most-common-passwords,9486.html>.
- [41] Dragon Research Group. *DRG SSH Username and Password Authentication Tag Clouds*. Accessed 16/09/2010. 2010. URL: <http://www.dragonresearchgroup.org/insight/sshpwauth-cloud.html>.
- [42] Jan Harrington. *Network Security, A Practical Approach*. Morgan Kaufmann, 2005.

- [43] Josh Harriman. *A Testing Methodology for Rootkit Removal Effectiveness*. Accessed 15/01/2011. Dublin, Ireland, 2007. URL: http://www.symantec.com/avcenter/reference/testing_methodology_for_rootkit_removal.pdf.
- [44] Guillaume Delugré. *Reversing the Broacom NetExtreme's firmware*. Accessed 25/11/2010. 2010. URL: <http://esec-lab.sogeti.com/dotclear/index.php?post/2010/11/21/Presentation-at-Hack.lu:-Reversing-the-Broacom-NetExtreme-s-firmware>.
- [45] Codenomicon. *Codenomicon Test Tools*. Accessed 21/05/2011. URL: <http://www.codenomicon.com>.
- [46] Ben Reardon. *VoIP honeynet: Observations of the VoIP Pilot Thus Far*. Accessed 17/06/2010. 2009. URL: http://honeynet.org.au/?q=honeynet_part2.
- [47] Craig Valli. "An Analysis of Malfeasant Activity Directed at a VoIP Honeypot". In: *Proceedings of the 8th Australian Digital Forensics Conference*. Perth, Western Australia: secAU, Edith Cowan University, Western Australia, 2010, pp. 162–167.
- [48] Ben Reardon. *Sunday (sundayddr) SIP scanning worm. When printers turn bad*. Accessed 01/12/2010. 2010. URL: http://honeynet.org.au/?q=sunday_scanner.
- [49] Sandro Gauci. *Distributed SIP scanning during Halloween weekend*. Accessed 02/12/2010. URL: <http://blog.sipvicious.org/2010/11/distributed-sip-scanning-during.html>.
- [50] P. Zimmermann. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. Accessed 02/12/2010. 2010. URL: <http://tools.ietf.org/html/draft-zimmermann-avt-zrtp-22>.
- [51] H. Sengar et al. "VoIP intrusion detection through interacting protocol state machines". In: *International Conference on Dependable Systems and Networks (DSN'06)* (2006), pp. 393–402.

- [52] S. Kent and R. Atkinson. *RFC2406: IP Encapsulating Security Payload*. Accessed 15/01/2011. 1998. URL: <http://tools.ietf.org/html/rfc2406>.
- [53] *OpenVPN*. Accessed 15/01/2011. 2010. URL: <http://openvpn.net/index.php/open-source.html>.
- [54] Scott Charney. *Collective Defense: Applying Public Health Models to the Internet*. Accessed 15/01/2011. 2010. URL: <http://go.microsoft.com/?linkid=9746317>.
- [55] Bruce Schneier. *Internet Quarantines*. Accessed 15/12/2010. 2010. URL: <http://www.schneier.com/blog/archives/2010/11/quarantine.html>.
- [56] M.a. Zissman. "Comparison of four approaches to automatic language identification of telephone speech". In: *IEEE Transactions on Speech and Audio Processing* 4.1 (Jan. 1996), pp. 31–44.
- [57] Sandro Gauci. *SIPVicious*. Accessed 19/08/2010. URL: <http://code.google.com/p/sipvicious/>.
- [58] Jon Oberheide et al. "Virtualized in-cloud security services for mobile devices". In: *Proceedings of the First Workshop on Virtualization in Mobile Computing*. ACM, 2008, pp. 31–35.
- [59] A.R.A. Grégio et al. "Malware distributed collection and pre-classification system using honeypot technology". In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. Vol. 7344. 2009, p. 10.
- [60] Martin Renwanz and Mathias Bohge. *A realistic VoIP traffic generation and evaluation tool for OMNeT++*. Accessed 02/12/2010. 2010. URL: <http://www.tkn.tu-berlin.de/research/omnetVoipTool/>.
- [61] *OMNeT++*. Accessed 16/09/2010. 2010. URL: <http://www.omnetpp.org>.
- [62] *INET for OMNeT++*. Accessed 16/09/2010. 2010. URL: <http://inet.omnetpp.org>.

- [63] Mathias Bohge and Martin Renwanz. *VoIP Tool for OMNeT++*. Accessed 16/09/2010. URL: <http://www.tkn.tu-berlin.de/research/omnetVoipTool/>.
- [64] Mathias Bohge and M. Renwanz. "A realistic VoIP traffic generation and evaluation tool for OMNeT++". In: *First International OMNeT++ Workshop*. Gent, Belgium, 2008.
- [65] Michael Tüxen, Irene Rüngeler, and Erwin Rathgeb. "Interface connecting the INET simulation framework with the real world". In: *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems* (2008).
- [66] Michael Tüxen, Irene Rüngeler, and Erwin Rathgeb. "Integration of SCTP in the OMNeT++ Simulation Environment". In: *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems* (2008).
- [67] Ugo Maria Colesanti, Carlo Crociani, and Andrea Vitaletti. "On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed". In: *Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks - PE-WASUN '07* (2007), p. 25.
- [68] Scapy. *Scapy*. Accessed 12/08/2010. URL: <http://www.secdev.org/projects/scapy/>.
- [69] Bryan Burns et al. *Security Power Tools*. 1st ed. O'Reilly, 2007.
- [70] *AsteriskNOW*. Accessed 21/12/2010. 2010. URL: <http://www.asterisk.org/asterisknow/>.
- [71] Markus Koetter. *Dionaea*. Accessed 21/05/2010. URL: <http://dionaea.carnivore.it>.
- [72] Mohamed Nassar, Rodrigo do Carmo, and Pablo Masri. *Artemisa*. Accessed 21/05/2010. URL: <http://artemisa.sourceforge.net>.

- [73] Brettsg. *Statistics Don't Lie... Or Do They?* Accessed 05/11/2010. 2010. URL: <http://blog.tllo.com/2010/11/03/statistics-dont-lie-or-do-they/>.
- [74] Kelly Jackson Higgins. *Zeus Attackers Deploy Honeybot Against Researchers, Competitors*. Accessed 15/11/2010. 2010. URL: <http://www.darkreading.com/insiderthreat/security/attacks/showArticle.jhtml?articleID=228200070>.
- [75] WG Halfond, Jeremy Viegas, and Alessandro Orso. "A classification of SQL injection attacks and countermeasures". In: *International Symposium on Secure Software Engineering*. 2006.
- [76] Python. *Python*. Accessed 25/06/2010. URL: <http://www.python.org>.
- [77] Stefan Behnel, Robert Bradshaw, and Dag Sverre Seljebotn. *Cython*. Accessed 14/12/2010. 2010. URL: <http://www.cython.org>.
- [78] SEC. Accessed 19/08/2010. URL: <http://simple-evcorr.sourceforge.net/>.
- [79] Julio Casal, Dominique Karg, and Alberto Román. *OSSIM*. Accessed 19/10/2010. 2010. URL: <http://www.alienvault.com/community.php?section=Home>.
- [80] IView. *Cyberoam iView*. Accessed 19/10/2010. 2010. URL: <http://www.cyberoam-iview.org/>.
- [81] Mohammed Salem and Helen Armstrong. "Identifying DOS Attacks Using Data Pattern Analysis". In: *Australian Information Security Management Conference*. 2008, p. 55.
- [82] B. Campbell et al. *RFC3428: Session Initiation Protocol (SIP) Extension for Instant Messaging*. Accessed 15/01/2011. 2002. URL: <http://www.ietf.org/rfc/rfc3428.txt>.

- [83] Sean-Paul Correll and Luis Corrons. *The Economy of Rogueware: Analysis of the New Style of Online Fraud*. Accessed 26/12/2010. 2010. URL: <http://ain.ua/goto/http://www.pandasecurity.com/img/enc/TheBusinessofRogueware.pdf>.
- [84] Rainer Bye et al. "Application-level simulation for network security". In: *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–10.
- [85] Voip-Info.org. *Asterisk cmd SendText: Send a text message*. Accessed 23/12/2010. 2010. URL: <http://www.voip-info.org/wiki/view/Asterisk+cmd+SendText>.
- [86] MaxMind. *GeoIP City*. Accessed 18/12/2010. 2010. URL: http://www.maxmind.com/app/locate_demo_ip.

A. SEC Rules

A.1. Scan Attack Correlation

```
# Collecting information on certain events (normalisation)
# SIP scan (any type)
type=Single
ptype=RegExp
pattern=^.+\\.+.+\\. SIP (\\S+) scan
    \\[\\*\\*\\.+.+(\\d+\\/\\d+)-(\\S+)\\s+(\\S+)\\s+>\\s+(\\S+)
desc=SIP $1 SCAN FROM $4
action=event $1_SCAN_ON_$2_FROM_$3: $0; add SCAN_REPORT %t: %s

type=Single
ptype=RegExp
pattern=(\\S+)_SCAN_ON_(\\S+)_FROM_(\\S+):
context=!CONTEXT_$1_SCAN_ON_$2
desc=New $1 scan on $2
action=create CONTEXT_$1_SCAN_ON_$2 60; event SIP_SCAN_FROM_$3

# Flooding: more than x SIP scans in given window
type=SingleWithThreshold
window=3
thresh=6
context=!FLOODING
ptype=RegExp
pattern=SIP_SCAN_FROM_(\\S+)
desc=Threshold of scans from $1 exceeded
action=create FLOODING 3;
    write sec.log %t SIP scans attack detected

# Flooding on different hosts:
# more than x hosts scanned in given window
type=SingleWithThreshold
window=3
thresh=3
context=!FLOODING
ptype=RegExp
pattern=SIP_SCAN_FROM_(\\S+)
```

```
desc=Threshold of scans from $1 on different hosts exceeded
action=create FLOODING 3;
    write sec.log %t SIP scans attack detected
```

A.2. Crash Correlation

```
# Ignore phone crashes (shutdowns) when phone DEREGISTERS
type=Single
ptype=RegExp
pattern=DEREGISTER {\S+} (\S+) -> (\S+)
desc=PHONE $2 deregistered
action=create IGNORE_PHONE_$2 60

# Crash report (any host)
type=Single
ptype=RegExp
pattern=^(\d+-\d+-\d+) (\d+:\d+:\d+) (\S+) is down
desc=PHONE $3 DOWN
action=event PHONE_$3_DOWN: $0; add CRASH_REPORT %t: %s

# Crash on any host: ignore same host in the future
type=Single
pytype=RegExp
pattern=PHONE_(\S+)_DOWN:
context=!CRASH_FROM_$1_CONTEXT
desc=Phone $1 crashed
action=create CRASH_FROM_$1_CONTEXT 60; event PHONE_DOWN: $0

# Phone crash reports from different hosts
type=SingleWithThreshold
window=60
thresh=2
ptype=RegExp
pattern=PHONES_DOWN:
context=!IGNORE_PHONE_CRASHES_CONTEXT
desc=Phone crash report threshold exceeded
action=write sec.log %t: %s;
    shellcmd ssh heartbeat@localhost echo UPDATE >> /tmp/update_report
```

B. Proof-of-concept SIP Notification Scapy Script (Python)

```
def createMessage(dst, msg):
    s = "MESSAGE sip:1100@" + dst + " SIP/2.0\n"
    s += "Via: SIP/2.0/UDP 192.168.50.100:5060\n"
    s += "From: \"server\" <sip:server@192.168.50.100:5060>\n"
    s += "To: \"alice\" <sip:1100@" + dst + ">\n"
    s += "Call-ID: 12345\n"
    s += "CSeq: 1 MESSAGE\n"
    s += "Content-Type: text/plain\n"
    s += "Content-Length: %i\n\n" % len(msg)
    s += msg

    return IP(dst=dst)/UDP(dport=5060, sport=5060)/s

send(createMessage("192.168.50.11", "User notification!"))
```

C. Simulation Code (C++)

The following code sections are the relevant parts to the core functionality of the network simulation. Initialisation, shut down, and similar sections of code are not shown.

C.1. User Agent Module

```
void UA::handleMessage(cMessage *msg)
{
    if (msg->isSelfMessage())
    {
        // The message sent to the module by itself should be the attack timer
        if (msg == &m_timer)
        {
            if (par("attackType").longValue() == 1)
            {
                // Record the number of scan packets sent for later analysis
                m_vectorAttackerScans.record(++m_numAttackerScanMessages);

                // Send a repeating scan attack to random users
                IPvXAddress victimAddress;
                AttackPacket* pk = this->generateScanAttackPacket(&victimAddress);

                m_statsVictims.collect(victimAddress.get4().getDByte(3));

                // Send to victim
                this->sendToUDP(pk, m_port, victimAddress, 5060);

                // Schedule next attack
                this->scheduleAt(simTime() + par("scanAttackDelay"), &m_timer);
            }
            else if (par("attackType").longValue() == 2)
            {
                // Record the number of exploit packets sent for later analysis
                m_vectorAttackerExploits.record(++m_numAttackerExploits);

                // Send an exploit attack to a random user
                IPvXAddress victimAddress;
```

```

        AttackPacket* pk = this->generateExploitAttackPacket(&victimAddress);

        m_statsVictims.collect(victimAddress.get4().getDByte(3));

        // Send to victim
        this->sendToUDP(pk, m_port, victimAddress, 5060);

        // Schedule next attack
        this->scheduleAt(simTime() + par("exploitAttackDelay"), &m_timer);
    }
}
else
{
    // Do not react to incoming messages if phone crashed due to
    // exploit attack
    if (!m_phoneDown) this->handleIncomingMessage(msg);

    delete msg;
}
}

void UA::handleIncomingMessage(cMessage *msg)
{
    SipPacket *sip = dynamic_cast<SipPacket*>(msg);
    AttackPacket *attack = dynamic_cast<AttackPacket*>(msg);
    if (sip != NULL)
    {
        // User notification
        if (strcmp(sip->getMsg(), "MESSAGE") == 0)
        {
            this->bubbleNw("User notification MESSAGE");
        }

        // OPTIONS ping from RGW
        else if (strcmp(sip->getMsg(), "OPTIONS") == 0)
        {
            SipPacket *pk = this->generateSipPacket();
            pk->setMsg("200 OK");
            pk->setSeqNo(sip->getSeqNo());
            pk->setSeqMethod(sip->getSeqMethod());
            pk->setTo(sip->getFrom());
            pk->setFrom(sip->getTo());
            this->sendToUDP(pk, m_port, m_rgwAddress, 5060);
        }
    }
}

```



```

else if (attack != NULL)
{
    EV << "User under attack: ";
    switch (attack->getType())
    {
        case 0: EV << "dummy"; break;
        case 1: this->onAttackScan(attack); break;
        case 2: this->onAttackExploit(attack); break;
        case 3: this->onAttackFlooding(attack); break;
    }
    EV << endl;
}
}

```

C.2. Residential Gateway Module

```

void RGW::handleMessage(cMessage *msg)
{
    if (msg->isSelfMessage())
    {
        if (msg == &m_timer)
        {
            this->bubbleNw("Ping response timeout - UA down");
            m_vectorPingsTimeout.record(++m_numPingsTimeout);

            // Report timeout to SEC
            this->sendHeartbeatReport();
            this->cancelEvent(&m_pingTimer);
        }
        else if (msg == &m_pingTimer)
        {
            // Send one OPTIONS ping to UA
            this->sendPing();
            m_vectorPingsSent.record(++m_numPingsSent);

            // Schedule next ping timer
            this->scheduleAt(simTime() + par("heartbeatInterval").doubleValue(),
                            &m_pingTimer);

            // Schedule timeout timer in case UA is down
            this->scheduleAt(simTime() + 1, &m_timer);
        }
    }
    else
    {

```

```

        this->handleIncomingMessage(msg);
    }
}

void RGW::handleSipMessage(SipPacket *sip)
{
    // Response to OPTIONS ping?
    if(strcmp(sip->getMsg(), "200 OK") == 0 &&
        strcmp(sip->getSeqMethod(), "OPTIONS") == 0)
    {
        this->bubbleNw("Received pong");
        m_vectorPingResponsesRcvd.record(++m_numPingResponsesRcvd);
        this->cancelEvent(&m_timer);
    }
    else
    {
        this->forwardToUA(sip);
    }
}

void RGW::handleAttack(AttackPacket *attack)
{
    m_vectorAttacks.record(++m_numAttacks);

    // Scan attack packet
    if (attack->getType() == 1)
    {
        m_vectorIdsDetections.record(++m_numIdsDetections);
        m_datacollector->recordIdsDetection();

        // Get source IP address from network layer
        IpWithBlacklist *mIp = this->getIpModule();
        IPAddress ip = mIp->getLastSourceAddress();

        this->bubbleNw("IDS detected scanning activity");
        this->sendSnortReport(ip.str().c_str());
    }

    // Exploit attack packet
    else if (attack->getType() == 2)
    {
        // Check if signature is in database
        for (VectorSigIt it = m_knownExploitSignatures.begin();
            it != m_knownExploitSignatures.end(); it++)
        {
            EV << "Checking signature \"" << *it << "\" against " <<

```

```

        "payload \"" << attack->getPayload() << "\"" << endl;

// Compare attack signature to database of known signatures
if(strcmp(attack->getPayload(), *it) == 0)
{
    this->bubbleNw("Exploit signature match");

    m_vectorExploitSignatureHits.record(++m_numExploitSignatureHits);
    m_datacollector->recordExploitSignatureHits();

    delete attack;
    attack = NULL;

    break;
}

// Exploit signature has not been detected
if (attack != NULL)
{
    m_vectorExploits.record(++m_numExploits);
    m_datacollector->recordExploits();
}

// Forward all non-blocked attacks to UA
if (attack != NULL) this->forwardToUA(attack);
}

```

C.3. Event Correlation Module

```

void SEC::handleMessage(cMessage *msg)
{
    if (!msg->isSelfMessage())
    {
        // Incoming message is either a Snort or a crash (heartbeat) report
        ReportPacket *report = dynamic_cast<ReportPacket*>(msg);
        if (report != NULL)
        {
            const int numUsers = this->getAncestorPar("numUsers").longValue();

            // IDS report
            if (report->getType() == 0)
            {
                this->getParentModule()->bubble("Received IDS report");
            }
        }
    }
}

```

```

m_vectorScanReports.record(++m_numScanReports);

if (m_numScanReports == par("scanReportsThreshold").longValue())
{
    this->getParentModule()->bubble("Scan report limit reached");

    m_numScanReports = 0;

    // Send IP blacklist update to all users
    for (int i = 0; i < numUsers; i++)
    {
        UpdatePacket *update = new UpdatePacket();
        update->setType(0);

        char buf[16];
        snprintf(buf, 16, "%i",
            IPAddress(report->getPayload()).getInt());
        update->setPayload(buf);
        update->setName("Backlist update");

        snprintf(buf, 16, "users[%d].rgw", i);
        this->sendToUDP(PK(update), m_port,
            IPAddressResolver().resolve(buf), 5060);
    }
}

// Heartbeat report
else if (report->getType() == 1)
{
    this->getParentModule()->bubble("Received heartbeat report");
    m_vectorCrashReports.record(++m_numCrashReports);

    if (m_numCrashReports == par("crashReportsThreshold").longValue())
    {
        // Send signature update to all users
        for (int i = 0; i < numUsers; i++)
        {
            // Create update packet for IDS/firewall on RGW
            UpdatePacket *update = new UpdatePacket();
            update->setType(1);
            update->setPayload("Kphone vulnerable");
            update->setName("IDS update");

            char buf[16];
            snprintf(buf, 16, "users[%d].rgw", i);

```

```

        this->sendToUDP(PK(update), m_port,
            IPAddressResolver().resolve(buf), 5060);
    }
}
}
}

delete msg;
}
}

SipPacket* SEC::generateUserNotification()
{
    SipPacket* sip = new SipPacket();

    sip->setFrom("unknown <sip:>");
    sip->setTo("ua[0] <sip:100@10.0.0.1>");
    sip->setSeqNo(1);
    sip->setSeqMethod("MESSAGE");
    sip->setMsg("MESSAGE");

    std::string s("User notification text");
    sip->setContentLength(s.length());
    sip->setBody(s.c_str());

    return sip;
}

```

D. Heartbeat Monitor

```
void ssh_send(const char *msg)
{
    if (msg == NULL) error("ERROR: invalid argument");

    // Get current timestamp
    char timestamp[32];
    time_t secs;
    struct tm *loctime;

    secs = time(NULL);
    loctime = localtime(&secs);
    strftime(timestamp, 32, "%Y-%m-%d %H:%M:%S", loctime);

    char ssh_cmd[1024];
    snprintf(ssh_cmd, 1024,
        "ssh %s@%s echo \"%s %s is down >> /tmp/heartbeat_report\"",
        SSH_USER, SSH_HOST, timestamp, PHONE_ID);
    system(ssh_cmd);
}

int failed_response(int *failed)
{
    // Increase failed counter and check against THRESHOLD
    (*failed)++;
    if (*failed >= THRESHOLD)
    {
        fprintf(stderr,
            "Heartbeat failed %d times -> client down\n", *failed);
        ssh_send("Heartbeat Failed");
        *failed = 0;
    }
}

int main(int argc, const char *argv[])
{
    int sockfd, bytes, failed;
    char msg[512];
    char buffer[1024];
```

```

struct sockaddr_in addr;
struct hostent *server;

failed = 0;

// Construct OPTIONS message
strcat(msg, "OPTIONS sip:100@127.0.0.1 SIP/2.0\n");
strcat(msg, "Via: SIP/2.0/UDP 127.0.1.1:5061;branch=z9hG4bK;rport\n");
strcat(msg, "Content-Length: 0\n");
strcat(msg, "From: \"heartbeat\"<sip:100@1.1.1.1>;tag=c0a8320313c4\n");
strcat(msg, "Accept: application/sdp\n");
strcat(msg, "User-Agent: heartbeat\n");
strcat(msg, "To: \"sipphone\"<sip:100@1.1.1.1>\n");
strcat(msg, "Contact: sip:100@127.0.0.1:5061\n");
strcat(msg, "CSeq: 1 OPTIONS\n");
strcat(msg, "Call-ID: 289318289127391823\n");
strcat(msg, "Max-Forwards: 70\n\n");

// Get SIP server address
server = gethostbyname(HOST);
if (server == NULL) error("ERROR, no such host");

// Set address structure
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
memcpy(&addr.sin_addr.s_addr, server->h_addr, server->h_length);
addr.sin_port = htons(PORT);

// Create socket
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) error("ERROR opening socket");

// Connect to phone
if (connect(sockfd, (struct sockaddr*)&addr, sizeof(addr)) < 0)
{
    error("ERROR connecting to phone");
}

while (1)
{
    // Send ping
    bytes = write(sockfd, msg, strlen(msg));
    if (bytes < 1) error("ERROR sending message");

    // Receive pong (this is where an unopen SIP port is noticed)
    bytes = read(sockfd, buffer, 256);
}

```

```

        if (bytes < 0)
        {
            failed_response(&failed);
        }
        else
        {
            failed = 0;

            // Print response
            printf("RESONSE:\n%s\n", buffer);
        }

        // Wait 10 seconds until next ping
        sleep(PING_INTERVAL);
    }

    // Close socket
    close(sockfd);

    return 0;
}

```


E. Honeybot

E.1. Configuration

```
sip = {
    use_authentication = "yes"
    domain = "localhost"
    port = "5060"
    user = "100"
    useragent = "softphone"
    secret = "F2DS13G5"
    record_rtp = "yes"
}

services = {
    serve = ["sip"]
}
```

E.2. Relevant VoIP Module Code (Python)

```
def parseSdpMessage(msg):
    """Parses an SDP message (string), returns a tuple of dictionaries
    with {type: value} entries: (sessionDescription, mediaDescriptions)"""
    # Normalize line feed and carriage return to \n
    msg = msg.replace("\n\r", "\n")

    # Sanitize input: remove superfluous leading and trailing newlines
    # and spaces
    msg = msg.strip("\n\r\t ")

    # Split message into session description, and media description parts
    SEC_SESSION, SEC_MEDIA = range(2)
    curSection = SEC_SESSION
    sessionDescription = {}
    mediaDescriptions = []
    mediaDescriptionNumber = -1
```

```

# Process each line individually
if len(msg) > 0:
    lines = msg.split("\n")
    for line in lines:
        # Remove leading and trailing whitespaces from line
        line = line.strip('\n\r\t ')

        # Get first two characters of line and check for "type="
        if len(line) < 2:
            raise SdpParsingError("Line too short")
        elif line[1] != "=":
            raise SdpParsingError("Invalid SDP line")

        type = line[0]
        value = line[2:].strip("\n\r\t ")

        # Change current section if necessary
        # (session -> media -> media -> ...)
        if type == "m":
            curSection = SEC_MEDIA
            mediaDescriptionNumber += 1
            mediaDescriptions.append({})

        # Store the SDP values
        if curSection == SEC_SESSION:
            if type not in sessionDescriptionTypes:
                raise SdpParsingError(
                    "Invalid session description type: " + type)
            else:
                sessionDescription[type] = value
        elif curSection == SEC_MEDIA:
            if type not in mediaDescriptionTypes:
                raise SdpParsingError(
                    "Invalid media description type: " + type)
            else:
                mediaDescriptions[mediaDescriptionNumber][type] = value

    return (sessionDescription, mediaDescriptions)

def parseSipMessage(msg):
    """Parses a SIP message (string), returns a tuple (type, firstLine,
    header, body)"""
    # Sanitize input: remove superfluous leading and trailing newlines
    # and spaces
    msg = msg.strip("\n\r\t ")

```

```

# Split request/status line plus headers from body: we don't care
# about the body in the SIP parser
parts = msg.split("\n\n", 1)
if len(parts) < 1:
    logger.warn("SIP message is too short")
    raise SipParsingError("SIP message is too short")

msg = parts[0]

# Python way of doing a ? b : c
body = len(parts) == 2 and parts[1] or ''

# Normalize line feed and carriage return to \n
msg = msg.replace("\n\r", "\n")

# Split lines into a list, each item containing one line
lines = msg.split('\n')

# Get message type (first word, smallest possible one is "ACK" or "BYE")
sep = lines[0].find(' ')
if sep < 3:
    raise SipParsingError("Malformed request or status line")

msgType = lines[0][:sep]
firstLine = lines[0][sep+1:]

# Done with first line: delete from list of lines
del lines[0]

# Parse header
headers = {}
for i in range(len(lines)):
    # Take first line and remove from list of lines
    line = lines.pop(0)

    # Strip each line of leading and trailing whitespaces
    line = line.strip("\n\r\t ")

    # Break on empty line (end of headers)
    if len(line.strip(' ')) == 0:
        break

# Parse header lines
sep = line.find(':')
if sep < 1:
    raise SipParsingError("Malformed header line (no ':')")

```

```

# Get header identifier (word before the ':')
identifier = line[:sep]
identifier = identifier.lower()

# Check for valid header
if identifier not in shortHeaders.keys() and \
    identifier not in longHeaders.keys():
    raise SipParsingError("Unknown header type: {}".format(identifier))

# Get long header identifier if necessary
if identifier in longHeaders.keys():
    identifier = longHeaders[identifier]

# Get header value (line after ':')
value = line[sep+1:].strip(' ')

# The Via header can occur multiple times
if identifier == "via":
    if identifier not in headers:
        headers["via"] = [value]
    else:
        headers["via"].append(value)

# Assign any other header value directly to the header key
else:
    headers[identifier] = value

# Return message type, header dictionary, and body string
return (msgType, firstLine, headers, body)

class SipSession:
    def __authenticate(self, headers):
        global g_sipconfig

        if not g_sipconfig['use_authentication']:
            logger.debug("Skipping authentication")
            return

        logger.debug("'Authorization' in SIP headers: {}".format(
            'authorization' in headers))

    def sendUnauthorized(nonce):
        msgLines = []
        msgLines.append('SIP/2.0 ' + RESPONSE[UNAUTHORIZED])
        msgLines.append("Via: " + self.__sipVia)

```

```

msgLines.append("Max-Forwards: 70")
msgLines.append("To: " + self.__sipTo)
msgLines.append("From: " + self.__sipFrom)
msgLines.append("Call-ID: {}".format(self.__callId))
msgLines.append("CSeq: " + headers['cseq'])
msgLines.append("Contact: " + self.__sipContact)
msgLines.append("User-Agent: " + g_sipconfig['useragent'])
msgLines.append('WWW-Authenticate: Digest ' + \
    'realm="{}@{}",'.format(g_sipconfig['user'],
        g_sipconfig['domain']) + \
    'nonce="{}"'.format(nonce))
self.send('\n'.join(msgLines))

if "authorization" not in headers:
    # Calculate new nonce for authentication based on current time
    nonce = hash("{}".format(time.time()))

    # Send 401 Unauthorized response
    sendUnauthorized(nonce)

    raise AuthenticationError("Request was unauthenticated")

else:
    # Check against config file
    authMethod, authLine = headers['authorization'].split(' ', 1)
    if authMethod != 'Digest':
        logger.warn("Authorization method is not Digest")
        raise AuthenticationError("Method is not Digest")

    # Get Authorization header parts (a="a", b="b", c="c", ...) and put
    # them in a dictionary for easy lookup
    authLineParts = [x.strip(' \t\r\n') for x in authLine.split(',')]
    authLineDict = {}
    for x in authLineParts:
        parts = x.split('=')
        authLineDict[parts[0]] = parts[1].strip(' \n\r\t"\'')

    logger.debug("Authorization dict: {}".format(authLineDict))

    if 'nonce' not in authLineDict:
        logger.warn("Nonce missing from authorization header")
        raise AuthenticationError("Nonce missing")

    if 'response' not in authLineDict:
        logger.warn("Response missing from authorization header")
        raise AuthenticationError("Response missing")

```

```

# The calculation of the expected response is taken from
# Sipvicious (c) Sandro Gaucchi
realm = "{}@{}".format(g_sipconfig['user'], g_sipconfig['domain'])
uri = "sip:" + realm
a1 = hash("{}: {}: {}".format(
    g_sipconfig['user'], realm, g_sipconfig['secret']))
a2 = hash("INVITE: {}".format(uri))
expected = hash("{}: {}: {}".format(a1, authLineDict['nonce'], a2))

logger.debug("a1: {}".format(a1))
logger.debug("a2: {}".format(a2))
logger.debug("expected: {}".format(expected))

# Report authentication incident
i = incident("dionaea.modules.python.sip.authentication")
i.authenticationSuccessful = expected == authLineDict['response']
i.realm = realm
i.uri = uri
i.nonce = authLineDict['nonce']
i.challengeResponse = authLineDict['response']
i.expected = expected
i.report()

if expected != authLineDict['response']:
    sendUnauthorized(authLineDict['nonce'])
    raise AuthenticationError("Authorization failed")

logger.info("Authorization succeeded")

```